AIX バージョン 7.2

パフォーマンス・マネージメント

AIX バージョン 7.2

パフォーマンス・マネージメント

IBM

お願い

本書および本書で紹介する製品をご使用になる前に、 505 ページの『特記事項』 に記載されている情報をお読みください。

本書は AIX バージョン 7.2 および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原典： AIX Version 7.2
Performance management

発行： 日本アイ・ビー・エム株式会社

担当： トランスレーション・サービス・センター

目次

本書について

この資料は、アプリケーション・プログラマー、保守エンジニア、システム・エンジニア、システム管理者、経験を積んだエンド・ユーザー、およびシステム・プログラマーに、プロセッサー、ファイル・システム、メモリー、ディスク入出力、Network File System (NFS)、Java、および通信入出力のパフォーマンスの評価および調整などのタスクの実行方法に関するまとまった情報を提供します。この資料では、システムおよびアプリケーションのインプリメンテーションを含め、それらの効率的な設計についても扱っています。この資料は、オペレーティング・システムに付属のドキュメンテーション CD にも収録されています。

強調表示

本書では、以下の強調表示規則を使用します。

太字	名前がシステムによって事前定義されているコマンド、サブルーチン、キーワード、ファイル、構造体、ディレクトリー、および他の項目を示します。 さらに、ユーザーが選択するボタン、ラベル、およびアイコンなどのグラフィカル・オブジェクトも示します。
イタリック	実際の名前または値をユーザーが提供する必要があるパラメーターを示します。
モノスペース	具体的なデータ値の例、表示される可能性があるテキストの例、プログラマーとして作成する可能性があるプログラム・コードの一部の例、システムからのメッセージ、またはユーザーが実際に入力する必要がある情報を示します。

AIX における大/小文字の区別

AIX® オペレーティング・システムでは、すべてケース・センシティブとなっています。これは、英大文字と小文字を区別するということです。 例えば、**ls** コマンドを使用するとファイルをリストできます。 `LS` と入力すると、システムはそのコマンドが `not found` (見つかりません) と応答します。 同様に、**FILEA**、**FiLea**、および **filea** は、同じディレクトリーにある場合でも、3 つの異なるファイル名です。 予期しない処理が実行されないように、常に正しい大/小文字を使用するようにしてください。

ISO 9000

当製品の開発および製造には、ISO 9000 登録品質システムが使用されました。

パフォーマンス・マネージメント

このトピックでは、アプリケーション・プログラマー、保守エンジニア、システム・エンジニア、システム管理者、経験を積んだエンド・ユーザー、およびシステム・プログラマーを対象として、プロセッサー、ファイルシステム、メモリー、ディスク入出力、NFS、Java および通信入出力のパフォーマンスの評価とチューニングなどの作業を実行する方法を詳しく説明します。この資料では、効率的なシステムとアプリケーションの設計、およびその実装についても説明しています。この資料はオペレーティング・システムに同梱されているドキュメンテーション CD にも収納されています。

注**: lparstat**、**vmstat**、**iostat**、**mpstat** などの統計情報ツール (Perfstat アプリケーション・プログラム・インターフェース (API) またはシステム・パフォーマンス測定インターフェース (SPMI) API に基づくアプリケーションを含む) によって報告される各メトリックは、任意の時点で同じサンプリング間隔で並行して実行される場合でも、ある程度は異なります。

パフォーマンス・マネージメントの新機能

「パフォーマンス・マネージメント」資料に関する新規情報または著しく変更された情報を読み、理解してください。

新規情報または変更情報の参照方法

この PDF ファイルでは、新規情報および変更情報の左端にリビジョン・バー (|) が付いている場合があります。

2017 年 4 月

- 490 ページの『ディスクおよびディスク・アダプターのチューナブル・パラメーター』のトピックで、ファイバー・チャネル・アダプターの未処理要求の制限チューナブル・パラメーターに関する情報が追加されました。

2016 年 12 月

- 78 ページの『スレッド環境変数』のトピックで、*AIXTHREAD_SCOPE* 環境変数に関する情報が更新されました。
- AIX 6.1 以降ではサポートされないことから、`mempools` チューナブル・パラメーターに関する情報が削除されました。

2016 年 10 月

以下の情報は、「パフォーマンス調整」トピック・コレクションに対して行われた更新の要約です。

- 192 ページの『VMM スレッド中断オフロード』のトピックに関する情報が追加されました。

パフォーマンスの基本

システム・パフォーマンスを評価するには、プログラム実行のダイナミックスを理解している必要があります。

システム・ワークロード

システムのワークロードを正確かつ完全に定義することは、そのパフォーマンスを予測し、理解する上で重要です。

ワークロードの差は、CPU クロック・スピードまたはランダム・アクセス・メモリー (RAM) サイズの差より、システムのパフォーマンスの測定結果にはるかに多くの変化をもたらすことがあります。 ワークロードの定義には、システムに送信される要求のタイプや速度だけでなく、実行する予定の正確なソフトウェア・パッケージや社内のアプリケーション・プログラムも含める必要があります。

システムがバックグラウンドで実行している処理を含めることは重要です。例えば、ユーザーのシステムに NFS マウントのファイルシステムが含まれていて、他のシステムがそれに頻繁にアクセスする場合、ユーザーのシステムが公式にはサーバーでないとしても、それらのアクセスの処理はワークロード全体の重要な部分と考えられます。

異なるシステム間での比較が行えるように標準化されたワークロードのことをベンチマーク といいます。ただし、実際のワークロードでベンチマークのアルゴリズムと環境を再現したものは、ほとんどありません。 本来実際のアプリケーションから引き出された業界標準のベンチマークでさえ、さまざまな種類のハードウェア・プラットフォームに移植できるように、単純化され、均質化されています。業界標準のベンチマークの唯一の有効な使用法は、重要な評価の対象となる候補システムの範囲を絞ることです。 したがって、システムのワークロードやパフォーマンスを理解しようとするときに、ベンチマークの結果のみに頼るべきではありません。

ワークロードは、以下のカテゴリーに分類できます。

マルチユーザー
: 個々の端末を通じて作業を実行依頼する、複数のユーザーからなるワークロード。 一般的には、そのようなワークロードのパフォーマンス目標は、指定された最悪の場合の応答時間を保持する一方でシステム・スループットを最大化すること、または一定のワークロードについて可能な最善の応答時間を確保することのいずれかにあります。

サーバー
: 他のシステムによる要求からなるワークロード。 例えば、ファイル・サーバー・ワークロードは主としてディスクの読み取り/書き込み要求です。 サーバーはマルチユーザー・ワークロード (および NFS またはその他の入出力アクティビティー) のディスク入出力コンポーネントなので、所定の応答時間制限内の最大スループットとして同じ目標が適用されます。 その他のサーバー・ワークロードは、数値計算プログラム、データベース・トランザクション、プリンター・ジョブなどからなっています。

ワークステーション
: システムのキーボードを通じて作業を実行依頼し、ディスプレイで結果を受け取る、シングル・ユーザーからなるワークロード。 一般的には、そのようなワークロードの最も優先順位の高いパフォーマンス目標は、ユーザーの要求に対する応答時間を最小化することです。

パフォーマンス目標

システムが処理する必要のあるワークロードを定義した後で、パフォーマンス基準を選択し、その基準を基にしてパフォーマンス目標を設定することができます。

コンピューター・システムの総合的パフォーマンス基準は、応答時間およびスループットです。

応答時間 は、要求が実行依頼されてから、その要求からの応答が戻されるまでの経過時間です。 例えば、以下のものがあります。

- データベースの照会にかかる時間
- 文字を端末にエコーするのにかかる時間
- Web ページにアクセスするのにかかる時間

スループット は、一定時間内に達成される作業量の測定基準です。 例えば、以下のものがあります。

- 1 分間のデータベース・トランザクション
- 1 秒間のファイル転送サイズ (キロバイト)
- 1 秒間のファイル読み書きサイズ (キロバイト)
- 1 分間の Web サーバー・ヒット数

これらのメトリック間の関係は複雑です。 時には、応答時間を犠牲にしてより高いスループットが得られる場合もあれば、スループットを犠牲にして応答時間が向上する場合もあります。 また状況によって、1 つ変更するだけで両方が改善されることもあります。許容されるパフォーマンスが得られるのは、適度なスループットと適度な応答時間の組み合わせを基にした場合です。

システムの計画またはチューニングの際には、指定したワークロードを処理するときに、応答時間とスループットの両方について、必ず明確な目標を持つようにしてください。明確な目標がないと、2 番目に重要なシステム・パフォーマンスの 1 局面を改善するために、分析時間と資金を費やす危険を冒すことになります。

プログラム実行モデル

ワークロードのパフォーマンス特性を明確に調べるには、以下の図に示されているように、静的ではなく動的なプログラム実行のモデルを必要とします。

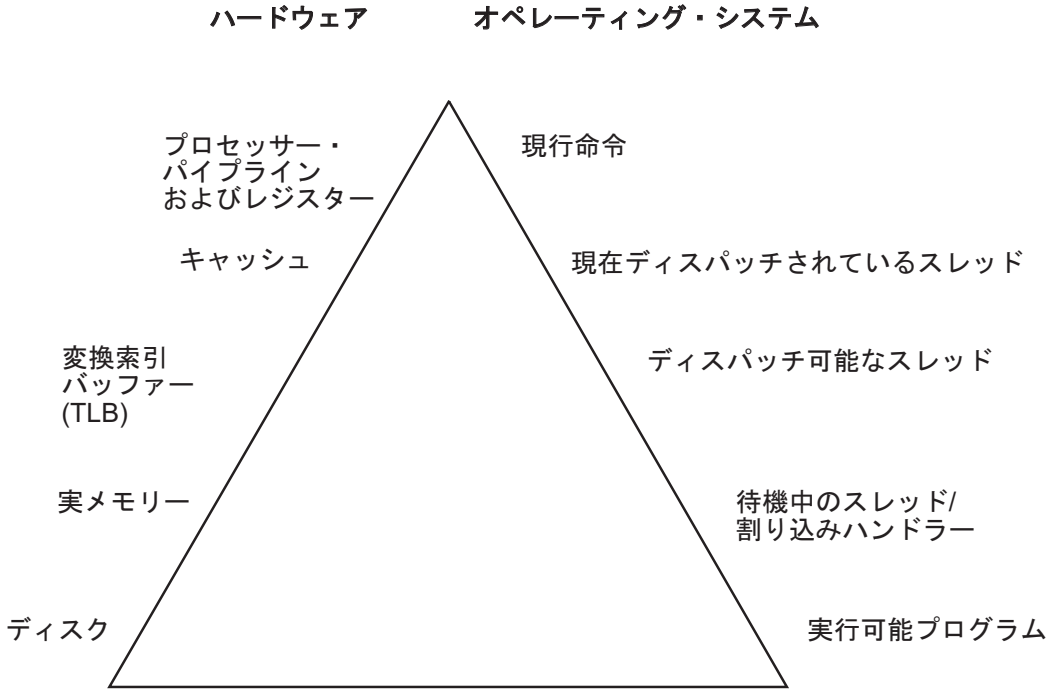


図 *1.* プログラム実行の階層： この図は、底辺を下にした三角形です。 左側はハードウェア・エンティティーを表し、該当する右側のオペレーティング・システムのエンティティーに対応しています。 プログラムは、最下位レベルのディスクに保管されている状態から、最上位レベルのプログラム命令を実行するプロセッサーに到達する必要があります。 例えば (下から上に)、ディスク・ハードウェア・エンティティーは実行可能プログラムを保管し、実メモリーはオペレーティング・システムの待ちスレッドと割り込みハンドラーを保持し、変換索引バッファー機構はディスパッチ可能スレッドを保持し、キャッシュは現在ディスパッチされているスレッドを格納し、プロセッサーのパイプラインとレジスターは現行命令を格納します。

プログラムを実行するためには、プログラムがハードウェア階層とオペレーティング・システム階層の両方を、並行して上っていく必要があります。 ハードウェア階層の各エレメントは、その下のエレメントより少なく、費用もかかります。 プログラムは、それぞれのリソースごとに他のプログラムに対処する必要があるだけでなく、あるレベルから次のレベルへの移行に時間もかかります。 プログラム実行のダイナミックスを理解するには、階層内の各レベルについて基本的な知識が必要です。

ハードウェア階層

通常、あるハードウェア・レベルから別のレベルに移動するのに要する時間は、主として低い方のレベルの待ち時間 (要求を出してから最初のデータを受信するまでの時間) と一致します。

ハード・ディスク

スタンドアロン・システムで実行中のプログラムの最も遅い操作は、ディスクからのコードまたはデータの取得です。これは、次の理由によります。

- 指定されたブロックにアクセスするようディスク・コントローラーに指示する必要がある (キューイング遅延)。
- ディスク・アームが正しいシリンダーにシークする必要がある (シーク待ち時間)。
- 読み取り/書き込みヘッドの下に正しいブロックが回ってくるまで待機する必要がある (回転待ち時間)。
- データをコントローラーに送信し (送信時間)、次にアプリケーション・プログラムに渡す必要がある (割り込み処理時間)。

プログラム内の明示的な読み取りまたは書き込み要求に加えて、低速なディスク操作が行われる原因は数多くあります。 システム・チューニング・アクティビティーとは、不要なディスク入出力を見つけ出すことであると言えます。

実メモリー

実メモリー (ランダム・アクセス・メモリーや RAM とも呼ばれる) はディスクよりも高速ですが、バイト当たりの費用はずっと高くつきます。 オペレーティング・システムは現在使用中のコードおよびデータのみを RAM に保持しようとし、余分なものをディスクに格納します (ただし、最初から余分なものを RAM に読み込むことはありません)。

RAM はプロセッサーと比較して必ずしも高速ではありません。 典型的な例では、ハードウェアが RAM アクセスの必要を認識する時点と、データまたは命令がプロセッサーで使用可能になる時点との間に、多数のプロセッサー・サイクルの RAM 待ち時間が発生します。

ディスク上に格納されている (またはまだ入っていない) 仮想メモリーのページに対してアクセスが行われようとする場合は、ページ・フォールトが起こり、そのページがディスクから読み込まれるまで、プログラムの実行は中断状態となります。

変換索引バッファー (TLB)

仮想メモリーのインプリメントによって、プログラマーはシステムの物理制限から分離されます。 プログラマーは、メモリーが非常に大きいかのようにプログラムを設計し、コーディングします。命令およびデータに対するプログラムの仮想アドレスを、RAM からその命令およびデータを入手するために必要な実アドレスに変換するのは、システムの責任範囲です。 このアドレス変換処理には時間がかかる場合があるので、システムは、最近アクセスされた仮想メモリー・ページの実アドレスを、変換索引バッファー (TLB) と呼ばれるキャッシュに保持します。

実行中のプログラムが小さなプログラムおよびデータ・ページにアクセスしている限り、仮想ページから実ページへの完全なアドレス変換を、RAM アクセスごとに再実行する必要はありません。 プログラムが *TLB* ミス と呼ばれる、TLB エントリーを持たない仮想メモリー・ページにアクセスしようとすると、アドレス変換を行うために、*TLB* ミス待ち時間 と呼ばれる多数のプロセッサー・サイクルが必要となります。

キャッシュ

プログラムの RAM 待ち時間回数を最小化するために、システムは命令およびデータ用のキャッシュを内蔵しています。 必要な命令またはデータが既にキャッシュ内にある場合、プロセッサーは次のサイクルでキャッシュ・ヒット結果、命令またはデータを遅延なしで使用することができます。 そうでなければ、キャッシュ・ミスによって、RAM 待ち時間が発生します。

システムによっては、2 つまたは 3 つのレベルのキャッシュがあり、通常それらを L1、L2、および L3 と呼びます。 特定のストレージへの参照が L1 ミスの結果に終わると、次に L2 がチェックされます。 L2 でミスが生じた場合は、参照は次のレベルに進みます。つまり、L3 が存在すれば L3 に進み、存在しなければ RAM を参照します。

キャッシュのサイズと構造はモデルによって異なりますが、その効率的な使用法の原理は同一です。

パイプラインとレジスター

パイプライン式の、スーパースカラー・アーキテクチャーを使用すると、ある特定の状況下では、複数命令の同時処理が可能になります。 汎用レジスターおよび浮動小数点レジスターの大きいセットがあるので、プログラムのデータのかなりの量を、頻繁に保管したり再ロードしたりするのではなく、レジスターに保持することができます。

最適化コンパイラーは、これらの能力を最大限利用するように設計されています。 運用プログラムを生成する際には、そのプログラムがどれほど小さくても、必ずコンパイラーの最適化機能を使用するようにしてください。 パフォーマンスを最大にするためのプログラムのチューニングについては、「*Optimization and Tuning Guide for XL Fortran, XL C and XL C++*」に解説されています。

ソフトウェア階層

プログラムを実行するにはさらに、プログラムがソフトウェア階層の一連のステップを上がっていかなければなりません。

実行可能プログラム

ユーザーがプログラムの実行を要求すると、オペレーティング・システムはいくつかの操作を行って、ディスク上の実行可能プログラムを実行プログラムに変換します。

まず、ユーザーの現行 *PATH* 環境変数内のディレクトリーをスキャンして、プログラムの正しいコピーを検出する必要があります。 次に、システム・ローダー (**ld** コマンドと混同しないでください。これはバインダーです) が、プログラムから共用ライブラリーへの外部参照をすべて解決しなければなりません。

ユーザーの要求を表すために、オペレーティング・システムは 1 つのプロセスを作成します。これは、専用仮想アドレス・セグメントなどの一組のリソースで、どの実行プログラムにも必須です。

オペレーティング・システムはさらに、そのプロセス内で自動的に単一スレッドを作成します。 スレッドとは、プログラムの単一インスタンスの現在の実行状態です。 AIX では、プロセッサーやその他のリソースへのアクセスは、プロセス・ベースではなくスレッド・ベースで割り当てられます。 アプリケーション・プログラムにより複数のスレッドをプロセス内に作成することができます。 これらのスレッドは、そのスレッドが実行されるプロセスが所有するリソースを共用します。

最後に、システムはプログラムのエントリー・ポイントに分岐します。 エントリー・ポイントを含むプログラム・ページが既にメモリー内にない場合 (プログラムが最近コンパイルされたか、実行されたか、またはコピーされた場合にそうなる可能性があります)、その結果起こるページ・フォールト割り込みが原因で、ページはその補助ストレージから読み取られます。

割り込みハンドラー

外部イベントが起こったことをオペレーティング・システムに通知するメカニズムは、現在実行中のスレッドを中断して割り込みハンドラーに制御権を移動するというものです。

割り込みハンドラーを実行する前に、必要なハードウェアの状態を保管して、割り込み処理の完了後、システムが確実にスレッドのコンテキストを復元できるようにする必要があります。 新たに起動された割り込みハンドラーは、ハードウェア階層を上へ移動する際の遅延をすべて経験します (ページ・フォールトを除く)。割り込みハンドラーがごく最近実行されたのでなければ (あるいは介入するプログラムが非常に小さくなければ)、そのコードまたはデータのどれかが TLB またはキャッシュ内に残っている見込みはありません。

中断されたスレッドが再度ディスパッチされると、その実行コンテキスト (レジスターの内容など) は論理的に復元されるので、スレッドは正しく機能します。ただし、TLB およびキャッシュの内容は、プログラムのその後の要求に基づいて再構成する必要があります。 したがって、割り込みハンドラーおよび中断されたスレッドの両方で、割り込みの結果かなりのキャッシュ・ミスおよび TLB ミスによる遅延が生じることがあります。

待ちスレッド

実行するプログラムが即時には実行できない要求、例えば同期入出力操作 (明示的またはページ・フォールトの結果として) を行うと、必ずそのスレッドはその要求が完了するまで待ち状態に置かれます。

通常、このために、要求そのものに必要な時間に加えて、 TLB およびキャッシュの待ち時間がもう一組生ずる結果となります。

ディスパッチ可能スレッド

スレッドがディスパッチ可能であるが、実行されていない場合、スレッドは実質的な仕事を何も達成していません。 さらに悪いことに、実行中のその他のスレッドによって、そのスレッドのキャッシュ線が再利用され、実メモリーのページが再利用される可能性があり、その結果、スレッドが最後にディスパッチされるとき、さらに遅延が生じることになります。

現在ディスパッチされているスレッド

スケジューラーは、プロセッサーの使用を最も強く要求するスレッドを選択します。

その選択に影響する考慮事項については、 43 ページの『プロセッサー・スケジューラーのパフォーマンス』に説明されています。 スレッドがディスパッチされると、プロセッサーの論理的な状態はスレッドが中断された時点に有効だった状態に復元されます。

現在のマシン・インストラクション

マシン・インストラクションの大部分は、TLB ミスまたはキャッシュ・ミスが生じていなければ、1 プロセッサー・サイクルで実行可能です。

対照的に、プログラムがそのプログラムの別の領域に速やかに分岐し、多数の異なる領域からのデータにアクセスして、 TLB およびキャッシュのミスを高率で引き起こす場合、実行される命令当たりのプロセッサー・サイクル (CPI) の平均数は、1 よりずっと大きくなる可能性があります。 そのプログラムは、参照の局所性に乏しいことを示しているといわれます。 プログラムは、そのジョブの実行には必要最小限の命令しか使用していないかもしれませんが、不必要に大きなサイクル数を消費しています。 1 つには命令数とサイクル数の間のこの不十分な相関のせいで、パスの長さを計算するためにプログラム・リストを検討しても、直接には時間的価値をもたらすことはありません。 通常は短いパスの方が長いパスより高速になりますが、速度の比率はパスの長さの比率とは大きく異なる場合があります。

コンパイラーは、非常に高度な方法でコードを再配置して、プログラムの実行に必要なサイクル数を最小化します。 最高のパフォーマンスを追求するプログラマーは、第一に、コンパイラーの最適化手法を結果論で修正しようとするのではなく、コンパイラーがコードの最適化を有効に行うのに必要な情報をすべて持っているかの確認に気を配る必要があります (『プリプロセッサーおよびコンパイラーの有効な使用法』を参照してください)。 最適化の効果を測る実際の尺度は、本当のワークロードのパフォーマンスです。

システム・チューニング

アプリケーション・プログラムを効率的にインプリメントした後、システムの全体的なパフォーマンスをさらに改善する作業は、システム・チューニングの問題になります。

システム・レベルのチューニングの対象となる主なコンポーネントは、次のとおりです。

通信入出力
: ワークロードのタイプおよび通信リンクのタイプに応じて、以下に示す 1 つ以上の通信デバイス・ドライバー、TCP/IP または NFS のチューニングが必要になる場合があります。

ハード・ディスク
: 論理ボリューム・マネージャー (LVM) は、ディスク上のファイルシステムおよびページング・スペースの配置を制御します。この配置によって、システムのシーク待ち時間に重大な影響を及ぼすことがあります。 ディスクのデバイス・ドライバーは、入出力要求を処理する順序を制御します。

実メモリー
: 仮想メモリー・マネージャー (VMM) は、フリーの実メモリー・フレームのプールを制御し、プールを再び満たすために、いつ、どこからフレームをスチールするかを決定します。

スレッドの実行
: スケジューラーは、次に制御を受け取るべきディスパッチ可能エンティティーを決定します。 AIX では、ディスパッチ可能なエンティティーがスレッドです。 43 ページの『スレッド・サポート』を参照してください。

パフォーマンス・チューニング

システムおよびワークロードのパフォーマンス・チューニングは、極めて重要です。

パフォーマンス・チューニング・プロセス

パフォーマンスのチューニングは、主としてリソース管理および正しいシステム・パラメーターの設定の問題です。

リソースを効率よく使用するためのワークロードおよびシステムのチューニングは、以下のステップからなっています。

1. システム上のワークロードを識別する。
2. 目標を設定する。
   a. 結果の測定方法を決定する。
   b. 目標を定量化し、優先順位を付ける。
3. システムのパフォーマンスを制限するクリティカル・リソースを識別する。
4. ワークロードのクリティカル・リソース所要量を最小化する。
   a. 選択の余地がある場合は、最も適切なリソースを使用する。
   b. 個々のプログラムまたはシステム機能のクリティカル・リソース所要量を削減する。
   c. リソースを並行して使用するために構造化する。
5. リソースの割り当てを、優先順位を反映するように変更する。
   a. 個々のプログラムの優先順位またはリソース制限を変更する。
   b. システム・リソース管理パラメーターの設定を変更する。
6. 目標が満たされるまで (またはリソースが飽和状態になるまで)、ステップ 3 からステップ 5 を繰り返す。
7. 必要に応じて、追加のリソースを適用する。

システム・パフォーマンス管理のフェーズごとに、適切なツールがあります (442 ページの『コマンドとサブルーチンのモニターとチューニング』を参照してください)。一部のツールは IBM® から入手可能です。その他のツールはサード・パーティーの製品です。 次の図は、単純な LAN 環境におけるパフォーマンス管理のフェーズを示しています。

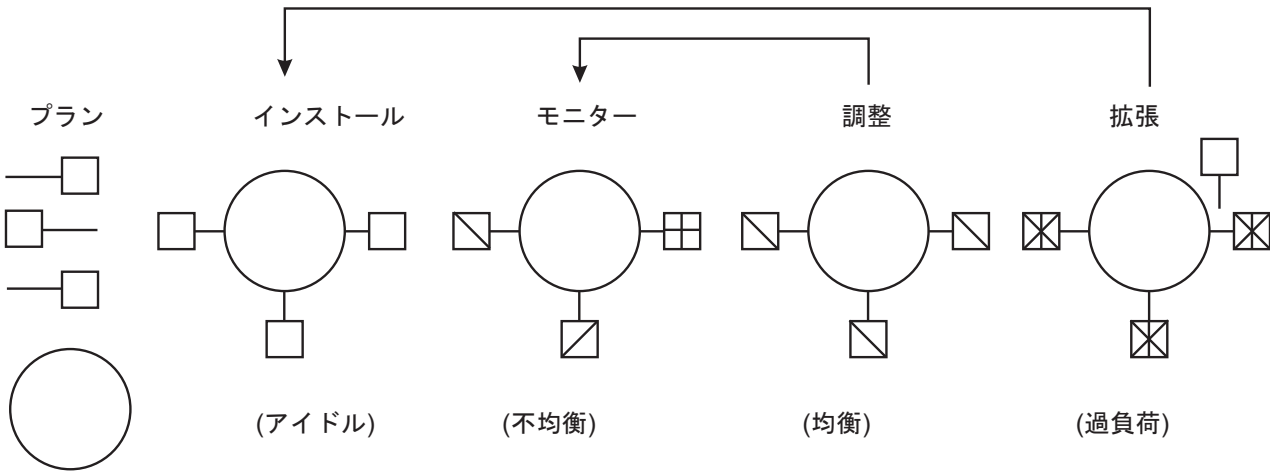


図 2. パフォーマンス・フェーズ： この図は、システムのパフォーマンス・チューニングの計画、インストール、モニター、チューニング、および拡張の各ステップを、おもりの付いた 5 つの円を使って表しています。 それぞれの円は、アイドル、不平衡、平衡、過負荷のパフォーマンス状態にあるシステムを表します。 基本的には、過負荷状態のシステムを拡張し、平衡がとれるまでシステムをチューニングし、不平衡のシステムをモニターし、拡張が必要なときには追加のリソースをインストールします。

ワークロードの識別

最も重要なことは、システムによって実行されるすべての作業を識別することです。 特に LAN 接続のシステムでは、複数のシステム間で相互にマウントされたファイルシステムが、各システムのユーザー間の非公式の合意だけで、簡単に実現してしまう可能性があります。 これらのファイルシステムを識別し、チューニング・アクティビティーの一部として考慮に入れる必要があります。

マルチユーザー・ワークロードでは、分析者は通常の要求率とピーク時の要求率の両方を定量化する必要があります。 また、ユーザーが実際に端末と対話する時間の配分について現実的であることも重要です。

この識別段階の重要なエレメントは、測定およびチューニング・アクティビティーを実動システムで実行する必要があるか、または別のシステムで (あるいは勤務時間外に) 実際のワークロードをシミュレートして達成できるかを判別することです。 分析者は、実稼働環境で得られる結果の確実性と、非実稼働環境での柔軟性とを比較検討する必要があります。非実稼働環境では、分析者は、性能低下またはそれ以上に悪い結果を招くリスクを伴う実験を行うことができます。

設定目標の重要性

測定可能な数量によって客観的に目標を設定することもできますが、実際に要求される結果は、満足のいく応答時間といった主観的なものである場合がよくあります。 さらに、分析者は、重要なものではなく、測定可能なものをチューニングしたいという誘惑に抵抗する必要があります。 システムが提供する測定法が望ましい改善に対応しない場合は、その測定法を工夫する必要があります。

目標を定量化することの最も価値のある局面は、達成すべき数値を選択することではなく、 (通常) 複数ある目標の相対的な重要性について共通の判断を下すことにあります。 これらの優先順位があらかじめ設定

され、関係するすべての人に理解されているのでない限り、分析者は、絶え間のない協議なしにはトレードオフの決定を下すことはできません。 分析者はまた、従来無視されてきたパフォーマンスの局面に対するユーザーや管理者の反応に驚くことになりがちです。 システムのサポートおよび使用が組織の境界を超えて行われている場合、パフォーマンス目標および優先順位について明確な共通の合意があることを確認するために、プロバイダーとユーザーの間で書面によるサービス・レベル・アグリーメント (SLA) が必要になる場合もあります。

クリティカル・リソースの識別

一般に、所定のワークロードのパフォーマンスは、 1 つまたは 2 つのクリティカル・システム・リソースの可用性およびスピードによって決まります。 分析者はそれらのリソースを正しく識別する必要があります。さもないと、永久に続く試行錯誤の作業に陥るリスクを冒すことになります。

システムには、実リソースと論理リソースの両方があり、さらに仮想リソースが存在する可能性もあります。 実リソースの使用率を評価するシステム・パフォーマンス・ツールは多数あるので、クリティカルな実リソースの識別は概して容易です。 パフォーマンスに影響を及ぼすことが多い実リソースは、次のとおりです。

- CPU サイクル
- メモリー
- 入出力バス
- 各種のアダプター
- ディスク・スペース
- ネットワーク・アクセス

論理リソースの識別はそれほど容易ではありません。 論理リソースは、一般に、実リソースをパーティションに分割するプログラミングの抽象概念です。 分割化は、実リソースを共用し、管理するために行われます。

POWER5 ベースの IBM System p システムで、Micro-Partitioning®、仮想シリアル・アダプター、仮想 SCSI および仮想イーサネットなどの仮想リソースを使用することができます。

実リソースおよび実リソース上に構築される論理リソースおよび仮想リソースの例を、以下にいくつか示します。

CPU

- プロセッサー・タイム・スライス
- CPU ライセンスまたは Micro-Partitioning
- 仮想イーサネット

メモリー

- ページ・フレーム
- スタック
- バッファー
- キュー
- テーブル
- ロックおよびセマフォー

ディスク・スペース

- 論理ボリューム
- ファイルシステム
- ファイル
- ロジカル・パーティション
- 仮想 SCSI

ネットワーク・アクセス

- セッション
- パケット
- チャネル
- 共用イーサネット

実リソースと同様に論理リソースおよび仮想リソースを認識することは重要です。 スレッドは、論理リソース不足で実行を阻止されることがあり、実リソース不足の場合とまったく同じです。また、ベースとなる実リソースを拡張しても、必ずしも追加の論理リソースの作成が保証されるわけではありません。例えば、保留状態の各 NFS リモート入出力要求を処理するには、サーバー上の NFS サーバー・デーモン、つまり **nfsd** デーモンが必要です。 したがって、**nfsd** デーモンの数によって、同時進行可能な NFS 入出力操作数が制限されます。 **nfsd** デーモン数の不足時は、システム計測機能によれば、各種の実リソース (CPU など) がごくわずかしか使用されていないことが示される場合があります。 実際は **nfsd** デーモン数が不足 (この不足が原因でその他のリソースを制約して使用できなくしている) している場合に、システムが十分に使用されておらず、しかも遅いという誤った印象を受ける可能性があります。 **nfsd** デーモンはプロセッサー・サイクルおよびメモリーを使用しますが、この問題を単に実メモリーを追加するか、またはより高速の CPU にアップグレードするだけで修正することはできません。 解決策としては、論理リソースとしての **nfsd** デーモンをもっと多く作成することです。

論理リソースおよびボトルネックは、アプリケーション開発中に不注意により作成されることがあります。データを渡すメソッドまたはデバイスを制御するメソッドにより、事実上、論理リソースが作成されます。そのようなリソースが偶然に作成された場合、一般にそれらの使用をモニターするツールはなく、その割り当てを制御するインターフェースもありません。 特定のパフォーマンス上の問題がその重要性を強調するまで、それらのリソースの存在は正しく認識されない可能性があります。

クリティカル・リソース所要量の最小化

ワークロードのクリティカル・リソース所要量を 3 つのレベルで最小化することを考慮してください。

適切なリソースの使用**:**

あるリソースを別のリソースに優先して使用する決定は、意識的に、特定のゴールを想定して行ってください。

アプリケーション開発中のリソースの選択の例として、メモリー使用量を増やして CPU 使用量を減らすというトレードオフがあります。 リソース選択を示す例として、よく行われるシステム構成上の決定に、ファイルをローカル側の個々のワークステーションに置くか、あるいはリモート側のサーバーに置くか、ということがあります。

クリティカル・リソースの所要量の削減**:**

ローカル側で開発されたアプリケーションの場合、同じ機能をより効率よく実行する方法、または不要な機能を除去する方法に関して、プログラムを検討することができます。

システム管理のレベルでは、クリティカル・リソースを求めて競合している優先順位の低いワークロードを、他のシステムに移動するか、別の機会に実行するか、またはワークロード・マネージャーによって制御することができます。

リソースの並列使用のための構造化**:**

ワークロードの実行には複数のシステム・リソースを必要とするので、リソースが分離していて並行して使用できるという事実を利用します。

例えば、オペレーティング・システムの先読みアルゴリズムは、プログラムがファイルに順次にアクセスするという事実を検出し、追加の順次読み取りを、アプリケーションによる直前のデータの処理と並行して行うようにスケジュールします。 並列性は、システム管理にも同様に適用されます。 例えば、アプリケーションが同時に複数のファイルにアクセスする場合、さらにディスク・ドライブを追加すると、同時にアクセスされるファイルが別々のドライブ上に置かれていれば、ディスク入出力速度が改善される可能性があります。

リソース割り当て優先順位

オペレーティング・システムには、アクティビティーの優先順位付けを行ういくつかの方法があります。

ディスク・ペーシングなど、一部はシステム・レベルで設定されます。 プロセス優先順位など、その他の優先順位は、個々のユーザーが、特定のタスクに指定した重要性を反映するように設定できます。

チューニング・ステップの繰り返し

パフォーマンス分析で明白なことは、常に次のボトルネックが存在することです。 あるリソースの使用を減らすということは、別のリソースがスループットまたは応答時間を制限することを意味します。

例えば、使用率レベルが以下のようなシステムがあるとします。

CPU: 90% ディスク: 70% メモリー: 60%

このワークロードは CPU 制約のワークロードです。 CPU のロードを 90% から 45% に減らすように、ワークロードを首尾よくチューニングできれば、パフォーマンスは 2 倍に向上するものと期待されるかもしれません。残念ながら、ワークロードには現在入出力の制約があり、その使用率はほぼ次のとおりです。

CPU: 45% ディスク: 90% メモリー: 60%

CPU 使用率の改善により、プログラムはディスク要求をより迅速に実行依頼することができますが、その後ディスク・ドライブの能力によって課される制限に突き当たることになります。 パフォーマンスの改善は、当初予想した 100% ではなく、30% にとどまるでしょう。

常に新しいクリティカル・リソースが存在します。 重要な問題は、手持ちのリソースでパフォーマンス目標を達成することができるかどうかです。

重要**:** **vmo**、**ioo**、**schedo**、**no**、および **nfso** チューニング・コマンドによって不適切なチューニングが行われると、システム・パフォーマンスやアプリケーション・パフォーマンスの低下、あるいはシステム・ハングなど、システムが思わぬ動作をすることがあります。変更は、パフォーマンス分析によってボトルネックが識別できたときだけ適用すべきです。

注**:** パフォーマンスがからむチューニング設定には、一般的な推奨事項のようなものはありません。

追加リソースの適用

前述のアプローチをすべて実行しても、システムのパフォーマンスがまだその目標に達しない場合は、クリティカル・リソースを強化または拡張する必要があります。

クリティカル・リソースが論理リソースで、基礎となる実リソースが十分にあれば、論理リソースは追加のコストなしに拡張することができます。 クリティカル・リソースが実リソースの場合は、分析者はさらに次のような問題を調べる必要があります。

- クリティカル・リソースがボトルネックとならないようにするには、そのクリティカル・リソースをどれほど強化または拡張する必要があるか。
- システムのパフォーマンスはその後その目標を達成するか、または別のリソースが先に飽和状態になるか。
- クリティカル・リソースが続発する場合、それらをすべて強化または拡張するのと、現在のワークロードを別のシステムに分割するのと、どちらが費用効果が高いか。

パフォーマンスのベンチマーク

あるソフトウェアの一部分のパフォーマンスを異なる環境で比較しようとすると、起こりうる多数のエラー (技術上のものもあれば、概念上のものもある) にさらされることになります。 このセクションでは、主として注意すべき事柄を説明します。 この資料のその他のセクションでは、経過時間およびプロセス固有の時間を測定するさまざまな方法について説明します。

システム・コールを処理する際に必要な経過時間を測ると、次のような数値が得られます。

- サービスを行うために命令が実行されていた実際の時間
- メモリーからの命令またはデータを待つ間、プロセッサーが停止した時間 (つまり、キャッシュ・ミスおよび TLB ミスのコスト)
- 呼び出しの開始および終了時にクロックにアクセスするために必要な時間
- システム・タイマー割り込みなどの定期的なイベントによって使用される時間
- 入出力割り込みなど、ランダムなイベントによって使用される時間

不正確な数値を報告しないように、ワークロードの測定は通常数回行います。 実際の処理時間には関係のない要因がすべて加算されるので、典型的な測定値の集りは、以下の図に示すような形式の曲線を描きます。

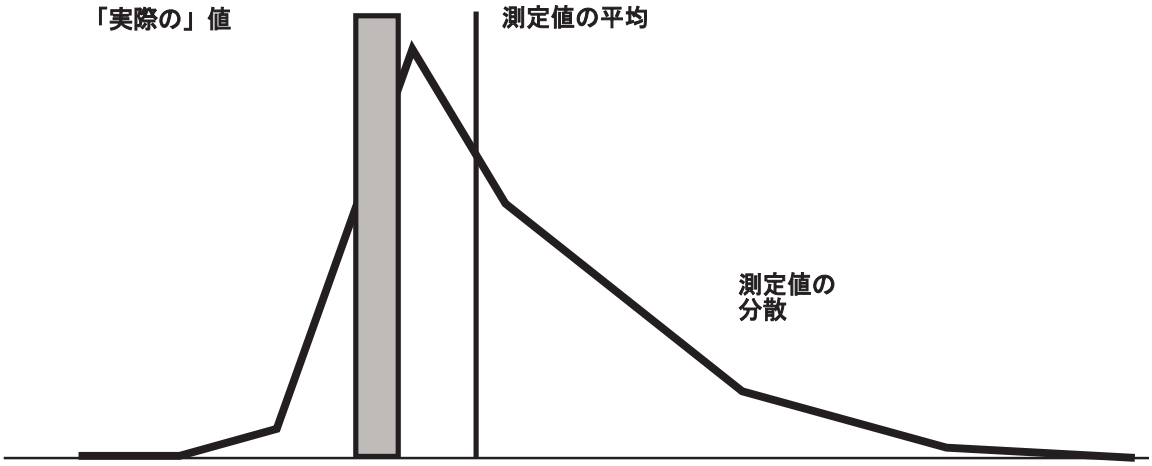


図 *3.* 測定の一般的なセットの曲線：

最下端は、確率の低い最適キャッシング状態を表しているか、丸め効果である可能性があります。

定期的に繰り返される無関係のイベントは、以下の図に示すような、二項 (最大値を 2 つ持つ) 曲線を描く可能性があります。

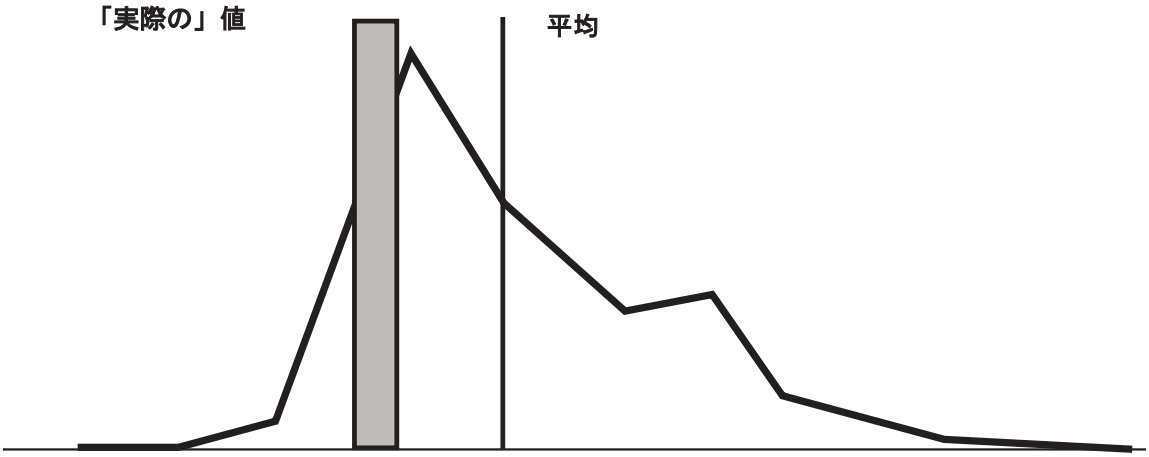


図 *4.* 二項曲線

1 つまたは 2 つの時間のかかる割り込みは、以下の図に示すように、さらに非対称の曲線を描く可能性があります。

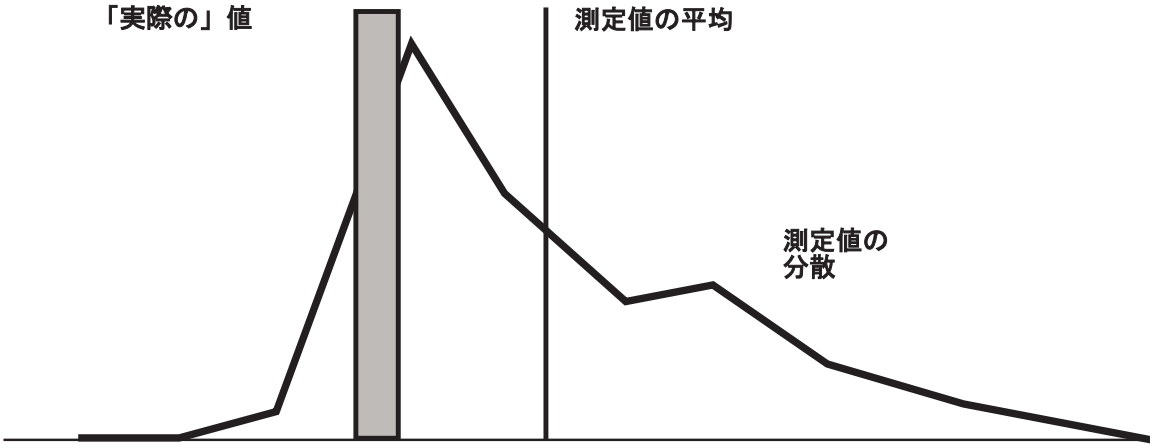


図 5. 非対称曲線

実際の値についての測定値の分布はランダムではなく、推論に基づく統計情報の標準的なテストを適用する場合は、最大限の注意を払う必要があります。 さらに、測定の目的によっては、平均値も実際の値もパフォーマンスの特徴を適切に示していない可能性があります。

システム・パフォーマンスのモニター

AIX は、パフォーマンス関連のシステム・アクティビティーをモニターするための多数のツールと手法を提供します。

連続システム・パフォーマンス・モニターの利点

システム・パフォーマンスを連続してモニターすると、幾つかの利点があります。

連続システム・パフォーマンス・モニターによって、以下の事柄が可能になります。

- 悪影響をもたらす前に潜在的な問題を検出できることがある
- ユーザーの生産性に影響を与える問題を検出できる
- 問題が初めて生じたときのデータを収集できる
- 比較のためのベースラインを確立できる

モニターを正常に行うには、以下の作業が必要です。

- オペレーティング・システムからのパフォーマンス関連情報の定期的な取得
- 問題診断における将来の利用のための情報の保管
- システム管理者に役に立つ情報の表示
- 追加のデータ収集を必要とする状態の検出、またはシステム管理者からのそのようなデータ収集の指示に対する応答、あるいはその両方
- 必要な明細データの収集および保管
- システムとアプリケーションに対する変更のトラッキング

コマンドを使用した連続システム・パフォーマンス・モニター

vmstat、**iostat**、**netstat**、および **sar** コマンドは、パフォーマンス・モニター機構を構築するための基礎になります。

シェル・スクリプトを作成して、コマンド出力上のデータ整理を行い、パフォーマンス上の問題について警告し、あるいは問題の発生時にシステムの状況に関するデータを記録することができます。 例えば、シェル・スクリプトは、CPU アイドルのパーセンテージがゼロの飽和状態になっていないか検査し、CPU が飽和状態になった場合は別のシェル・スクリプトを実行することができます。 以下のスクリプトは、スクリプトのユーザーが所有するプロセス以外で CPU 時間の大部分を消費した 15 のアクティブ・プロセスを記録します。

```
# ps -ef | egrep -v "STIME|$LOGNAME" | sort +3 -r | head -n 15
```

vmstat コマンドを使用した連続パフォーマンス・モニター

vmstat コマンドは、CPU、ページング、メモリーの使用状況の全体像をつかむために便利です。

以下は、 **vmstat** コマンドが生成するサンプル・レポートです。

```
# vmstat 5 2
kthr     memory             page              faults        cpu
----- ----------- ------------------------ ------------ -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 1  1 197167 477552   0   0   0   7   21   0 106 1114 451  0  0 99  0
 0  0 197178 477541   0   0   0   0    0   0 443 1123 442  0  0 99  0
```

vmstat コマンドからの最初のレポートは、最後のシステム・ブート以降の累積アクティビティーを示していることを覚えておいてください。 2 番目のレポートは、最初の 5 秒間のアクティビティーを示します。

vmstat コマンドの詳細については、 112 ページの『vmstat コマンド』、 139 ページの『vmstat コマンドによるメモリー使用量の判別』、および 197 ページの『vmstat コマンドによるディスク・パフォーマンスの評価』を参照してください。

iostat コマンドを使用した連続パフォーマンス・モニター

iostat コマンドは、ディスクおよび CPU の使用状況を判別するために便利です。

AIX オペレーティング・システムはディスク・アクティビティーのヒストリーを維持します。以下の例では、次のようなメッセージが表示されているため、ディスク入出力のヒストリーは使用不可にされています。

```
Disk history since boot not available.

The interval disk I/O statistics are unaffected by this.
```

ディスク入出力のヒストリーを使用可能にするには、コマンド・ラインから `smit chgsys` と入力し、「ディスク入出力のヒストリーを継続して維持する **(Continuously maintain DISK I/O history)**」フィールドで「はい **(true)**」を選択します。

iostat コマンドを実行すると、以下のようなレポートの例が表示されます。

```
# iostat 5 2

tty:      tin         tout   avg-cpu:  % user    % sys     % idle    % iowait
          0.1        102.3               0.5      0.2       99.3       0.1

               Disk history since boot not available.

               The interval disk I/O statistics are unaffected by this.

tty:      tin         tout   avg-cpu:  % user    % sys     % idle    % iowait
          0.2        79594.4              0.6      6.6       73.7      19.2

Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
```

```
hdisk1          0.0       0.0       0.0          0         0
hdisk0         78.2    1129.6     282.4       5648         0
cd1             0.0       0.0       0.0          0         0
```

iostat コマンドの最初のレポートは、ディスク・アクティビティー・カウンターの最後のリセット以降の累積アクティビティーを示しています。 2 番目のレポートは、最初の 5 秒間のアクティビティーを示します。

関連概念:

115 ページの『iostat コマンド』
iostat コマンドは、システムに、ディスク入出力の制約によるパフォーマンス上の問題があるかどうかに関して、最初の印象を得る最も速い方法です

関連タスク:

194 ページの『iostat コマンドによるディスク・パフォーマンスの評価』
評価は、システムのワークロードのピーク期間中、または入出力遅延を最小にする必要がある重要なアプリケーションの実行中に、interval パラメーターを指定して **iostat** コマンドを実行することによって開始します。

netstat コマンドを使用した連続パフォーマンス・モニター

netstat コマンドは、送信済み、受信済みのパケット数を判別するために便利です。

以下は、 **netstat** コマンドが生成するサンプル・レポートです。

```
# netstat -I en0 5
    input    (en0)     output           input   (Total)    output
 packets  errs  packets  errs colls  packets  errs  packets  errs colls
 8305067     0  7784711     0     0 20731867     0 20211853     0     0
       3     0        1     0     0        7     0        5     0     0
      24     0      127     0     0       28     0      131     0     0
CTRL C
```

netstat コマンドからの最初のレポートは、最後のシステム・ブート以降の累積アクティビティーを示していることを覚えておいてください。 2 番目のレポートは、最初の 5 秒間のアクティビティーを示します。

役に立つその他の **netstat** コマンド・オプションは、**-s** および **-v** です。詳細については、 325 ページの『netstat コマンド』を参照してください。

sar コマンドを使用した連続パフォーマンス・モニター

sar コマンドは、CPU の使用状況を判別するために便利です。

以下は、 **sar** コマンドが生成するサンプル・レポートです。

```
# sar -P ALL 5 2

AIX aixhost 2 5 00040B0F4C00    01/29/04

10:23:15 cpu    %usr    %sys    %wio   %idle
10:23:20  0        0       0       1      99
          1        0       0       0     100
          2        0       1       0      99
          3        0       0       0     100
          -        0       0       0      99
10:23:25  0        4       0       0      96
          1        0       0       0     100
          2        0       0       0     100
          3        3       0       0      97
          -        2       0       0      98
```

```
Average   0        2       0       0      98
          1        0       0       0     100
          2        0       0       0      99
          3        1       0       0      99
          -        1       0       0      99
```

sar コマンドは、最後のシステム・ブート以後の累積アクティビティーの報告を行いません。

sar コマンドの詳細については、 116 ページの『sar コマンド』および 198 ページの『sar コマンドによるディスク・パフォーマンスの評価』を参照してください。

topas コマンドを使用した連続システム・パフォーマンス・モニター

topas コマンドは、ローカル・システムにおけるアクティビティーに関する重要な統計情報 (実メモリー・サイズ、書き込みシステム・コール数など) を報告します。

topas コマンドは、80x25 文字ベースのディスプレイの表示に適応したフォーマットで、またはグラフィカル・ディスプレイで少なくとも同サイズのウィンドウに出力を表示するために、curses ライブラリーを使用します。 **topas** コマンドは、デフォルトでは 2 秒間隔でシステムから統計情報を抽出し、表示します。 **topas** コマンドは、以下の代替画面を表示します。

- 全システム統計情報
- 最もビジーなプロセスのリスト
- WLM 統計情報
- ホット物理ディスクのリスト
- ロジカル・パーティション・ディスプレイ
- クロス・パーティション・ビュー

topas コマンドを実行するには、`bos.perf.tools` ファイルセットおよび `perfagent.tools` ファイルセットをシステム上にインストールする必要があります。

topas コマンドの詳細については、*Commands Reference, Volume 5*の『topas コマンド』を参照してください。

全システム統計情報画面

全システム統計情報画面の出力は、1 つの固定セクションと 1 つの可変セクションで構成されています。

出力の左上の 2 行は、**topas** プログラムが実行されているシステムの名前、最終監視日時、およびモニターの間隔を表示します。 このセクションの下は可変セクションで、以下のサブセクションをリストします。

- CPU 使用率
- ネットワーク・インターフェース
- 物理ディスク
- WLM クラス
- プロセス

このセクションの右側は固定セクションで、以下の統計情報サブセクションを含んでいます。

- イベント/キュー (EVENTS/QUEUES)
- ファイル/TTY (FILE/TTY)
- ページング (PAGING)

- メモリー (MEMORY)
- ページング・スペース (PAGING SPACE)
- NFS

全システム統計情報画面のサンプル出力を以下に示します。

```
Topas Monitor for host:    aixhost             EVENTS/QUEUES    FILE/TTY
Wed Feb  4 11:23:41 2004   Interval:  2        Cswitch      53  Readch     6323
                                               Syscall     152  Writech     431
Kernel    0.0   |                            | Reads         3  Rawin         0
User      0.9   |                            | Writes        0  Ttyout        0
Wait      0.0   |                            | Forks         0  Igets         0
Idle     99.0   |############################| Execs         0  Namei        10
                                               Runqueue    0.0  Dirblk        0
Network  KBPS   I-Pack  O-Pack   KB-In  KB-Out  Waitqueue   0.0
en0       0.8     0.4     0.9     0.0     0.8
lo0       0.0     0.0     0.0     0.0     0.0  PAGING           MEMORY
                                               Faults        2  Real,MB    4095
Disk    Busy%     KBPS     TPS KB-Read KB-Writ  Steals        0  % Comp      8.0
hdisk0    0.0      0.0     0.0     0.0     0.0  PgspIn        0  % Noncomp  15.8
hdisk1    0.0      0.0     0.0     0.0     0.0  PgspOut       0  % Client   14.7
                                               PageIn        0
WLM-Class (Active)     CPU%    Mem%  Disk-I/O%  PageOut       0  PAGING SPACE
System                    0       0          0  Sios          0  Size,MB     512
Shared                    0       0          0                    % Used      1.2
Default                   0       0          0  NFS (calls/sec)  % Free     98.7
Name            PID CPU% PgSp Class          0  ServerV2       0
topas         10442  3.0  0.8 System           ClientV2       0   Press:
ksh           13438  0.0  0.4 System           ServerV3       0   "h" for help
gil            1548  0.0  0.0 System           ClientV3       0   "q" to quit
```

可変のプロセス・サブセクション以外では、列に基づいてすべてのサブセクションをソートできます。そのためには、対象の列の上部へカーソルを動かしてください。 プロセス・サブセクションを除くすべての可変サブセクションには、以下のビューがあります。

- トップ・リソース・ユーザーのリスト
- アクティビティーの合計を表す 1 行レポート

例えば、1 行レポート・ビューは、ディスクやネットワークの合計スループットを表示する場合があります。

CPU サブセクションでは、ユーザーは、使用中のプロセッサーのリストか、または上記の例に示されているように、グローバル CPU 使用率のいずれかを選択することができます。

topas モニターの最もビジーなプロセスをリストした画面

最もビジーなプロセスをリストした画面を表示するには、**topas** コマンドの **-P** フラグを使用します。

このスクリーンは全システム統計情報の画面に似ていますが、追加の詳細があります。 この画面もいずれかの列に基づいてソートできます。そのためには、対象の列の上部へカーソルを動かしてください。最もビジーなプロセスを示した画面の出力例を以下に示します。

```
Topas Monitor for host:    aixhost     Interval:   2    Wed Feb  4 11:24:05 2004

                                 DATA  TEXT  PAGE              PGFAULTS
USER       PID    PPID PRI NI    RES   RES SPACE    TIME CPU%  I/O  OTH COMMAND
root         1       0  60 20    202     9   202    0:04  0.0  111 1277 init
root       774       0  17 41      4     0     4    0:00  0.0    0    2 reaper
root      1032       0  60 41      4     0     4    0:00  0.0    0    2 xmgc
root      1290       0  36 41      4     0     4    0:01  0.0    0  530 netm
root      1548       0  37 41     17     0    17    1:24  0.0    0   23 gil
```

```
root      1806       0  16 41     4     0     4    0:00  0.0    0   12 wlmsched
root      2494       0  60 20     4     0     4    0:00  0.0    0    6 rtcmd
root      2676       1  60 20    91    10    91    0:00  0.0   20 6946 cron
root      2940       1  60 20   171    22   171    0:00  0.0   15  129 errdemon
root      3186       0  60 20     4     0     4    0:00  0.0    0  125 kbiod
root      3406       1  60 20   139     2   139    1:23  0.0   1542187 syncd
root      3886       0  50 41     4     0     4    0:00  0.0    0    2 jfsz
root      4404       0  60 20     4     0     4    0:00  0.0    0    2 lvmbb
root      4648       1  60 20    17     1    17    0:00  0.0    1   24 sa_daemon
root      4980       1  60 20    97    13    97    0:00  0.0   37  375 srcmstr
root      5440       1  60 20    15     2    15    0:00  0.0    7   28 shlap
root      5762       1  60 20     4     0     4    0:00  0.0    0    2 random
root      5962    4980  60 20    73    10    73    0:00  0.0   22  242 syslogd
root      6374    4980  60 20    63     2    63    0:00  0.0    2  188 rpc.lockd
root      6458    4980  60 20   117    12   117    0:00  0.0   54  287 portmap
```

topas モニターの WLM 統計情報画面

WLM 統計情報を示した画面を表示するには、**topas** コマンドの **-W** フラグを使用します。

この画面は以下のセクションに分割されています。

- 上部セクションは、全システム統計情報画面の WLM サブセクションに示されている、最もビジーな WLM クラスのリストで、やはり任意の列ごとにソート可能です。
- この画面の 2 番目のセクションは、矢印キーまたは *f* キーを使って選択した WLM クラスのホット・プロセスのリストです。

以下に示すのは、WLM フルスクリーン・レポートの例です。

```
Topas Monitor for host:    aixhost     Interval:   2    Wed Feb  4 11:24:29 2004
WLM-Class (Active)              CPU%      Mem%     Disk-I/O%
System                           0         0           0
Shared                           0         0           0
Default                          0         0           0
Unmanaged                        0         0           0
Unclassified                     0         0           0



================================================================================
                                 DATA  TEXT  PAGE               PGFAULTS
USER       PID    PPID PRI NI   RES   RES SPACE    TIME CPU%  I/O  OTH COMMAND
root         1       0  60 20   202     9   202    0:04  0.0    0    0 init
root       774       0  17 41     4     0     4    0:00  0.0    0    0 reaper
root      1032       0  60 41     4     0     4    0:00  0.0    0    0 xmgc
root      1290       0  36 41     4     0     4    0:01  0.0    0    0 netm
root      1548       0  37 41    17     0    17    1:24  0.0    0    0 gil
root      1806       0  16 41     4     0     4    0:00  0.0    0    0 wlmsched
root      2494       0  60 20     4     0     4    0:00  0.0    0    0 rtcmd
root      2676       1  60 20    91    10    91    0:00  0.0    0    0 cron
root      2940       1  60 20   171    22   171    0:00  0.0    0    0 errdemon
root      3186       0  60 20     4     0     4    0:00  0.0    0    0 kbiod
```

物理ディスク画面の表示

物理ディスクのリストを示す画面を表示するには、**topas** コマンドで **-D** フラグを使用します。

表示される物理ディスクの最大数は、**-d** フラグを指定してモニターされるホット物理ディスクの数です。ホット物理ディスク・リストは KBPS フィールドでソートされます。

以下は **topas -D** コマンドで生成されるレポートの例です。

```
Topas Monitor for host:    aixcomm    Interval:   2    Fri Jan 13 18:00:16 XXXX
================================================================================
Disk     Busy%  KBPS     TPS   KB-R   ART   MRT   KB-W   AWT   MWT   AQW    AQD
hdisk0     3.0  56.0     3.5   0.0    0.0   5.4   56.0   5.8   33.2  0.0    0.0
cd0        0.0   0.0     0.0   0.0   0.0    0.0   0.0    0.0    0.0  0.0    0.0
```

topas-D コマンドの詳細については、*Commands Reference, Volume 5*の『topas コマンド』を参照してください。

「クロス・パーティション (Cross-Partition)」パネルの表示

topas でクロス・パーティションの統計情報を表示するには、**topas** コマンドの **-C** フラグを使用するか、または他のパネルで **C** キーを押します。

この画面は以下のセクションに分割されています。

- 上部のセクションにはパーティション・セットからの集合データが表示されます。このデータは、全体のパーティション、メモリー、およびプロセッサー活動状況を表します。 **G** キーは、このセクションを要約リスト作成と詳細リスト作成とオフの切り替えに使用します。
- 下部のセクションにはパーティションごとの統計情報が表示されます。このセクションは、共用パーティションと専用パーティションの 2 つのセクションに分けて順に表示されます。 **S** キーは、共用パーティションのオン/オフの切り替えに使用します。 **D** キーは専用パーティションのオン/オフの切り替えに使用します。

下表は **topas -C** コマンド出力のフルスクリーンの例です。

```
Topas CEC Monitor            Interval:  10           Wed Mar  6 14:30:10 XXXX
Partitions     Memory (GB)               Processors
Shr:  4        Mon:  24  InUse:  14      Mon:  8  PSz:  4  Shr_PhysB:  1.7
Ded:  4        Avl:  24                  Avl:  8  APP:  4  Ded_PhysB:  4.1

Host         OS  M Mem InU Lp  Us Sy Wa Id  PhysB  Ent  %EntC Vcsw PhI
--------------------------------shared------------------------------------------
ptools1      A53 u 1.1 0.4  4  15  3  0 82   1.30  0.50  22.0  200   5
ptools5      A53 U  12  10  1  12  3  0 85   0.20  0.25   0.3  121   3
ptools3      A53 C 5.0 2.6  1  10  1  0 89   0.15  0.25   0.3   52   2
ptools7      A53 c 2.0 0.4  1   0  1  0 99   0.05  0.10   0.3  112   2
-------------------------------dedicated-------------------------------------
ptools4      A53 S 0.6 0.3  2  12  3  0 85   0.60
ptools6      A52   1.1 0.1  1  11  7  0 82   0.50
ptools8      A52   1.1 0.1  1  11  7  0 82   0.50
ptools2      A52   1.1 0.1  1  11  7  0 82   0.50
```

パーティションは、望ましい列にカーソルを移動して、ホスト、OS、OS M を除いていずれの列でもソートできます。

topas -C コマンドの詳細については、*Commands Reference, Volume 5*の『topas コマンド』を参照してください。

ローカルのロジカル・パーティション・レベル情報の表示

パーティション・レベル情報およびロジカル・プロセッサーごとのパフォーマンス・メトリックスを表示するには、**topas** コマンドで **-L** フラグを使用するか、または他のパネルで **L** キーを押します。

この画面は以下の 2 つのセクションに分割されています。

- 上部のセクションにはパーティション・レベル情報のサブセットが表示されます。
- 下部のセクションにはロジカル・プロセッサー・メトリックスのソートされたリストが表示されます。

以下は **topas -L** コマンドの出力例です。

```
Interval:   2            Logical Partition: aix       Sat Mar 13 09:44:48 XXXX
Poolsize: 3.0                 Shared SMT ON            Online Memory:   8192.0
Entitlement: 2.5               Mode: Capped            Online Logical CPUs:  4
                                                       Online Virtual CPUs:  2
%user  %sys  %wait  %idle physc  %entc %lbusy   app   vcsw  phint %hypv hcalls
 47.5  32.5    7.0   13.0   2.0   80.0  100.0   1.0    240    150   5.0   1500
================================================================================
logcpu minpf majpf  intr   csw  icsw runq lpa  scalls usr sys  wt idl  pc lcsw
cpu0    1135   145   134    78    60    2  95   12345  10  65  15  10 0.6  120
cpu1     998   120   104    92    45    1  89    4561   8  67  25   0 0.4  120
cpu2    2246   219   167   128    72    3  92   76300  20  50  20  10 0.5  120
cpu3    2167   198   127    62    43    2  94    1238  18  45  15  22 0.5  120
```

topas-L コマンドの詳細については、 *Commands Reference, Volume 5*の『topas コマンド』を参照してください。

topas/topasout/topasrec 用 SMIT パネル

SMIT パネルを使用すると、**topas** 記録機能とレポート生成を容易に構成およびセットアップすることができます。

topas smit パネルを表示するには、`smitty performance` (または `smitty topas`) と入力し、「**Configure Topas options (Topas** の構成オプション**)**」を選択します。

「Configure Topas Options (Topas の構成オプション)」メニューが表示されます。

```
                      Configure Topas Options
Move cursor to desired item and press Enter

 Add Host to topas external subnet search file (Rsi.hosts)
 List hosts in topas external subnet search file (Rsi.hosts)
 List active recordings
 Start new recording
 Stop recording
 List completed recordings
 Generate Report
 Setup Performance Management
```

詳しくは、*Commands Reference, Volume 5*の『topas コマンド』を参照してください。

topas 外部サブネット検索ファイル **(Rsi.hosts)** へのホストの追加**:**

PTX クライアントおよび **topas –C**|**topasrec –C** コマンドは、Remote Statistics Interface (Rsi) API がリモート・ホストの識別に使用される点において制限されています。

クライアントは、開始されるとすぐに、**inetd** デーモンの登録サービスである **xmquery** ポートで照会をブロードキャストします。リモート・ホストはこの照会を見て、**xmservd** または **xmservd** または **xmtopas** デーモンを開始するよう `inetd.conf` ファイルを構成し、照会しているクライアントに応答します。 既存のアーキテクチャーでは、**xmquery** 呼び出しは、照会を行っているシステムと同じサブネットにあるホスト内に限定されます。

この問題を回避するため、PTX はサブネットの外にある、ユーザーによってカスタマイズされたホスト・リストを常にサポートしてきました。 RSi はこのホスト・リスト (`RSi.hosts` ファイル) を読み取り、リストされているすべてのホスト名または IP を直接ポーリングします。 `RSi.hosts` ファイルはカスタマイズすることができます。デフォルトにより、RSi は優先順位に基づいて次のロケーションを検索します。

1. `$HOME/Rsi.hosts`
2. `/etc/perf/Rsi.hosts`
3. `/usr/lpp/perfmgr/Rsi.hosts`

このファイル・フォーマットは、次の例に示すように、インターネット・アドレス・フォーマットまたは完全修飾ホスト名のいずれかで、1 つのエントリー行ごとに 1 つのホストをリストします。

```
ptoolsl1.austin.ibm.com
9.3.41.206
...
```

「**Add Host to topas external subnet search file (Rsi.hosts) (topas** 外部サブネット検索ファイル **(Rsi.hosts)** にホストを追加**)**」オプションを選択して、ホストを **Rsi.hosts** ファイルに追加します。 「**List hosts in topas external subnet search file (Rsi.hosts)(topas** 外部サブネット検索ファイル **(Rsi.hosts)** 内のホストをリスト**)**」オプションを選択し、Rsi.hosts ファイル内のオプションのリストを表示します。

新規記録の開始**:**

ユーザーが選択した入力データに基づき、CEC/local の永続記録または非永続記録を開始するには、新規記録の開始を使用します。 ユーザーには、CEC/local の永続記録または非永続記録の開始用の別個のメニューが表示されます。

永続記録*:*

Persistent recordings (永続記録**)** は、SMIT から開始される記録であり、オプションで cut および retention を指定することができます。 各記録ファイルに何日分の記録を保管するか (cut)、および記録を削除するまで何日間保持するか (retention) を指定できます。 同一のタイプ (CEC またはローカル) の記録の **Persistent recording (**永続記録**)** の複数のインスタンスは、システム内で実行できません。 **Persistent recording (**永続記録**)** が開始されると、ユーザー指定オプションが指定された recording コマンドが起動されます。この永続記録により使用される、コマンド・ライン・オプションの同一のセットは、**inittab** に追加されます。 これで、システムのリブートまたは再始動時に記録は自動的に確実に開始されます。

Persistent local recording (永続ローカル記録**)** (バイナリー記録フォーマットまたは nmon 記録フォーマット) が既に実行しているシステムを考慮します。 ローカル・バイナリー記録の新規 **Persistent recording (**永続記録**)** を開始する場合は、最初に **Stop Recording (**記録の停止**)** オプションの下にある **Stop Persistent Recording (**永続記録の停止**)** オプションを使用して、既存の永続記録を停止する必要があります。 新規の **persistent local recording (**永続ローカル記録**)** は、**Start Persistent local recording (**永続ローカル記録の開始**)** オプションから開始する必要があります。 **Persistent recording (**永続記録**)** の開始は、同一記録フォーマットの永続記録がシステム内で既に実行中である場合、失敗します。 **Persistent recording (**永続記録**)** は **inittab** エントリーを追加するので、特権ユーザーのみが **Persistent recording (**永続記録**)** の開始を許可されます。

例えば、ファイル毎の保管の日数が **n** である場合、単一ファイルは最大 **n** 日分の記録を収容します。記録が **n** 日を超えた場合、新規のファイルが作成され、以降の記録はすべてこの新規ファイルに保管されます。ファイル毎の保管日数が **0** である場合、記録はただ 1 つのファイルに書き込まれます。 保持日数が **m** である場合、システムは最近の **m** 日以内に記録されたデータを持つ記録ファイルを保管します。 **topasrec** コマンドの同一記録インスタンスによって生成され、**m** 日より前に記録されたデータを持つ記録ファイルは、削除されます。

ファイル毎の保管日数のデフォルト値は、**1** です。

ファイル毎の保持日数のデフォルト値は、**7** です。

SMIT options for Start Recording (記録開始の SMIT オプション) メニューは、次のように表示します。

```
                  SMIT options for Start Recording

                         Start Recording
Move cursor to desired item and press Enter.

Start Persistent local Recording
Start Persistent CEC Recording
Start Local Recording
Start CEC Recording
```

永続的ローカル記録の開始*:*

ユーザーは永続的ローカル・バイナリー記録または nmon 記録のタイプを選択できます。

個別の記録を開始するには、Type of Persistent Recording メニューで「binary」または「nmon」を選択します。

```
               Type of Persistent Recording

Move cursor to desired item and press Enter.

   binary
   nmon



F1=Help                 F2=Refresh              F3=Cancel
```

バイナリー形式のレポートを選択した場合、そのレポートは次のように表示されます。

```
   Type or select values in entry fields.
   Press Enter AFTER making all desired changes.

  Type of Recording                                      binary
  Length of Recording                                   persistent
 Recording Interval in seconds                [300]
  * Number of Days to store per file      [1]
  * Number of Days to retain                      [7]
 Output Path                                                  []
 * Overwrite existing recording file     no
 * Enable WLE                                               no
 * Include Disk Basic Metrics                [Yes/No]
 * Include Service Time Metrics             [Yes/No]
 * Include Disk Adapter Basic Metrics  [yes/No]
 * Include Disk Adapter Service Time Metrics  [Yes/No]
```

記録間隔 (秒単位) は 60 の倍数でなければなりません。記録タイプがローカル・バイナリー記録である場合、ユーザーは SMIT 画面で IBM Workload Estimator (WLE) 報告書作成を使用可能にするオプションを持ちます。 WLE レポートは日曜日の 00:45 a.m. のみに生成され、レポート内のデータの整合性を保つためにはローカル・バイナリー記録を常に使用可能にしておく必要があります。 ローカル記録が常に使用可能になっている場合にのみ、週毎のレポートのデータが正しくなります。

生成された WLE レポートは、**/etc/perf/<hostname>_aixwle_weekly.xml** ファイルに保存されます。例えば、ホスト名が **ptoolsl1** の場合、週毎のレポートは **/etc/perf/ptoolsl1_aixwle_weekly.xml** ファイルに書き込まれます。

詳しくは、以下を参照してください。

- 23 ページの『永続記録』
- 使用可能な nmon フィルター

永続 ***CEC*** 記録の開始*:*

CEC 用の永続録を開始するには、「**start persistent CEC recording (**永続 **CEC** 記録の開始**)**」を使用します。 「**Final recording (**最終記録**)**」の開始は、それに続く画面で提供される入力データによって決まります。 入力画面は、最初にデフォルト値が記入された状態でロードされます。

記録間隔 (秒単位) は 60 の倍数でなければなりません。

詳しくは、 23 ページの『永続記録』を参照してください。

ローカル記録の開始*:*

start local recording (ローカル記録の開始**)** を使用して、その後の画面で提供される入力データに基づくローカル記録を開始します。ユーザーは、binary または nmon から選択でき、個別の記録を開始するために day (日)、hour (時間)、または custom (カスタム) を選択できます。

```
                            Type of Recording

Move cursor to desired item and press Enter.

  binary
  nmon

F1=Help                 F2=Refresh              F3=Cancel
```

binary または nmon の選択後、ユーザーは次の選択画面で day (日)、hour (時間)、または custom (カスタム) を選択する必要があります。

```
                           Length of Recording

Move cursor to desired item and press Enter.

  day
  hour
  custom

F1=Help                 F2=Refresh              F3=Cancel
F8=Image                F10=Exit                Enter=Do
```

day 記録または hour 記録の場合、記録間隔およびサンプル数は編集可能ではありません。 custom 記録の場合、記録間隔およびサンプル数は編集可能です。 記録間隔は 60 の倍数でなければなりません。 custom 記録の使用は、指定した間隔で、指定されたサンプル数のみを収集し、記録を終了するためです。 サンプル数がゼロと指定された場合、記録は停止されるまで連続的に実行されます。

画面に表示されるプリロード値は、デフォルト値です。

詳しくは、 26 ページの『NMON 記録』を参照してください。

CEC 記録の開始*:*

CEC のその後の画面の記録を開始するには、**start CEC recording (CEC** 記録の開始**)** を使用します。

それぞれの記録を開始するには、ユーザーは記録の長さ (day (日)、hour (時間)、または custom (カスタム)) を選択する必要があります。

```
                        Length of Recording

    Move cursor to desired item and press Enter.

     day
```

```
    hour
    custom

F1=Help                F2=Refresh              F3=Cancel
```

day 記録または hour 記録の場合、記録間隔サンプル数は編集可能ではありません。

custom 記録の場合、記録間隔およびサンプル数は編集可能であり、記録間隔は 60 の倍数でなければなりません。 custom 記録の使用は、指定した間隔で指定されたサンプル数のみを収集し、記録を終了するためです。 サンプル数がゼロと指定された場合、記録は停止されるまで連続的に実行されます。

***NMON* 記録:**

NMON は、**NMON recording (NMON 記録)** のカスタマイズに役立つ記録フィルターとともに出荷されます。 **NMON recording (NMON 記録)** の以下のセクションで選択または選択解除できます。

- JFS
- RAW kernel and LPAR
- ボリューム・グループ
- paging space
- MEMPAGES
- NFS
- WLM
- Large Page
- Shared Ethernet (for VIOS) Process
- Large Page and Asynchronous

注**:** 記録にディスク構成セクションが組み込まれている場合にのみ、回線毎のディスク、ディスク・グループ・ファイル、および希望するディスクは利用可能なオプションです。 記録にプロセス・リストが組み込まれている場合にのみ、プロセス・フィルター・オプションおよびプロセスしきい値オプションは利用可能です。

「プロセス・フィルターおよびディスク・フィルターは、同一ユーザーによる前回の記録で使用されたフィルター・オプションとともに、自動的にロードされます。 外部データ・コレクター開始または終了プログラムで、NMON 記録の開始または終了時に 外部 コマンドが起動されるように指定できます。 メトリックを記録するために外部コマンドを定期的に起動したい場合、それは外部データ・コレクター・スナップ・プログラムで指定できます。 NMON 記録の外部コマンドの使用について詳しくは、*Commands Reference, Volume 4* の nmon コマンドを参照してください。

命名規則**:**

記録されたファイルは、以下に示すように指定されたファイルに保管されます。

- ディレクトリーおよびファイル名接頭部を含むファイル名を与えられると、単一ファイル記録用の出力ファイルは次のとおりです。

スタイル	ファイル
ローカル Nmon スタイル:	<filename>_YYMMDD_HHMM.nmon
ローカル Nmon スタイル:	<filename>_YYMMDD_HHMM.topas
Topas スタイル CEC:	<filename>_YYMMDD_HHMM.topas

- ディレクトリーおよびファイル名接頭部を含むファイル名を与えられると、複数ファイル記録用の出力ファイル (カットおよび保持) は次のとおりです。

スタイル	ファイル
ローカル Nmon スタイル:	<filename>_YYMMDD.nmon
ローカル Nmon スタイル:	<filename>_YYMMDD.topas
Topas スタイル CEC:	<filename>_CEC_YYMMDD.topas

- ディレクトリーを含むがファイル名接頭部を含まないファイル名を与えられると、単一ファイル記録用の出力ファイルは次のとおりです。

スタイル	ファイル
ローカル Nmon スタイル:	<filename/hostname>_YYMMDD_HHMM.nmon
ローカル Nmon スタイル:	<filename/hostname>_YYMMDD_HHMM.topas
Topas スタイル CEC:	<filename/hostname>_CEC_YYMMDD_HHMM.topas

- ディレクトリーを含むがファイル名接頭部を含まないファイル名を与えられると、複数ファイル記録用の出力ファイル (カットおよび保持) は次のとおりです。

スタイル	ファイル
ローカル Nmon スタイル:	<filename/hostname>_YYMMDD.nmon
ローカル Nmon スタイル:	<filename/hostname>_YYMMDD.topas
Topas スタイル CEC:	<filename/hostname>_CEC_YYMMDD.topas

同一記録フォーマットで同一 **filename** パラメーター値 (デフォルト・ファイル名またはユーザー指定のファイル名) は、これら 2 つの記録プロセスが同一の記録ファイルに書き込む傾向があるので、同時には開始できません。

例:

1. ユーザーは、**/home/test/sample_bin** として指定された出力パスにローカル・バイナリー day(日) 記録を開始しようとしています。録音ファイルが 2008 年 3 月 10 日、時刻 12:05 に作成され、ホスト名が ses15 である場合、出力ファイル名は **/home/test/sample_bin/ses15_080310_1205.topas** です。
2. ユーザーが cut オプション 2、出力パスを **/home/test/sample** として指定した永続 CEC 記録を開始しようとしていたと想定します。録音ファイルが 2008 年 3 月 10 日、時刻 12:05 に作成され、ホスト名が ses15 であると想定すると、出力ファイル名は /home/test/sample_bin/ses15_cec_080310.topas です。 このファイル内で 2 日間分 (cut =2 のため) の記録が保管された後、3 月 12 日になると **/home/test/sample_bin/ses15_cec_080312.topas** と名付けられた記録ファイルが作成され、3 月 12 日と 3 月 13 日分の記録データが保管されます。

記録の停止:

現在実行中の記録を停止するには、「Stop recording (記録の停止)」を使用します。ユーザーは、リストからある特定の実行中の記録を選択し、それを停止できます。

メニューから、ユーザーは停止する記録のタイプを選択する必要があります。 記録のタイプの選択後、現在実行中の記録がメニューにリストされます。 ユーザーは、停止したい記録を選択できます。

以下に停止したい記録のタイプの選択用画面を示します。

```
                    Stop Recording

Stop persistent recording
Stop binary recording
Stop nmon recording
Stop CEC recording
```

注: 記録は、ユーザーが記録プロセスを停止するための必要権限を持っている場合にのみ停止できます。

活動中記録のリスト:

ユーザー指定ディレクトリー内のシステムで現在稼働中の記録をリストするには、 **List Active Recordings (**活動中記録のリスト**)** を使用します。

活動中記録をリストするには、次のようにします。

1. 記録のパスを入力します。
2. リストする記録のタイプを選択します。

```
                          Type of Recording
  Move cursor to desired item and press Enter.

  persistent
  binary
  nmon
  cec
  all

  F1=Help                 F2=Refresh              F3=Cancel
  F8=Image                F10=Exit                Enter=Do
  /=Find                  n=Find Next
```

これで活動中記録およびその指定されたパスの **Format, Start time, (**フォーマット、開始時刻、**)** および **Output path (**出力パス**)** がリストされます。

すべての永続記録の出力パスの前にアスタリスク (*) が付けられます。 WLE 対応の永続ローカル・バイナリー記録の場合、出力パスの前に番号記号 (#) が付けられます。

完了した記録のリスト:

ユーザー指定のディレクトリー・パス内の完了した記録のリストを表示するには、**List completed recordings (**完了した記録のリスト**)** を使用します。これらの完了した記録は、レポート・ファイルを生成するため、**Generate report menu (**レポートの生成メニュー**)** で使用できます。

完了した記録をリストするには、次のようにします。

1. 記録のパスを入力します。 これは記録ファイルを見つけるために使用されるパスです。
2. 使用される記録のタイプを選択します。

```
persistent
binary
nmon
cec
all
```

これで、指定パス内の完了した記録の **Recording Type, Start time (**記録タイプ、開始時刻**)** および **Stop time (**停止時刻**)** がリストされます。

既存の記録ファイルからのレポートの生成**:**

「**Generate Report (**レポートの生成**)**」オプションを使用して、ユーザーが指定したディレクトリー・パスの既存の記録ファイルからレポートを生成することができます。

選択したディレクトリー・パスが永続記録である場合、以下の条件が当てはまります。

1. 永続記録が実行中である場合、現在実行中の永続記録がレポートの生成用に選択されます。
2. 永続記録が実行中でない場合、直前に完了した永続記録がレポートの生成用に選択されます。

「**Generate Report (**レポートの生成**)**」 オプションを使用すると、レポートの生成に必要となる、記録ファイル、レポート・フォーマット、開始時刻、終了時刻、およびインターバルの値、さらにファイル名やプリンター名を入力するように求めるプロンプトが表示されます。

レポートを生成するには、次のステップを実行してください。

1. レポートを送信するファイル名またはプリンターを選択します。

```
                Send report to File/Printer

Move cursor to desired item and press Enter.

1 Filename
2 Printer
```

2. 記録ファイルを見つけるためのパスを選択します。

```
 Path to locate the recording file              [] +
```

3. (記録のタイプに基づき) レポート作成フォーマットを選択します。

```
* Reporting Format                              [] +
```

次に示すのは、コンマ区切り/スプレッドシート のレポート・タイプの例です。

```
* Type of Recording                             []
* Reporting Format                              []
    Type                                        [mean] +
  Recording File name                           []
* Output File                                   []
```

次に示すのは、*nmon* レポート・タイプの例です。

```
* Type of Recording                             []
* Reporting Format                              []
  Recording File name                           []
* Output File                                   []
```

注**:** 「**Output file (**出力ファイル**)**」フィールドは、コンマ区切り/スプレッドシート・タイプと *nmon* タイプでは必須ですが、他のすべてのレポート・フォーマットでは任意指定です。 **topas** 記録は、コンマ区切りとスプレッドシートのレポート・フォーマットの中間のタイプのみをサポートします。

次に示すのは、要約/ディスク要約/詳細/ネットワーク要約 レポート・タイプの例です。

```
* Type of Recording                              []
* Reporting Format                               []
  Begin Time (YYMMDDHHMM)                        []
  End Time (YYMMDDHHMM)                          []
  Interval                                       []
  Recording File name                            []
  Output File (defaults to stdout)               []
```

上記のすべての例で、最初の 2 つのフィールドは変更不可であり、直前の選択からの値が入ります。

レポート出力としてプリンターが選択されている場合、「**Output File (**出力ファイル**)**」フィールドは、システムで構成されているプリンターのリストからの必須の「**Printer Name (**プリンター名**)**」フィールドで置換されます。

```
* Printer Name                                  [ ] +
```

CEC 記録およびローカル・バイナリー記録で使用可能なレポート・フォーマットについて詳しくは、*Commands Reference, Volume 5* の topas コマンドを参照してください。

Setup Performance Management:

このメニューは、パフォーマンス・マネージメントのセットアップと構成に使用されます。

```
  Setup Performance Management

Move cursor to desired item and press Enter.

  Enable PM Data Transmission
  Disable PM Data Transmission
  Retransmit Recorded Data
  Change/Show Customer Information
  Change/Show Data Retention Period
  Change/Show Trending Days and Shift Timing
```

- **Enable PM Data Transmission**

  パフォーマンス・データを Electronic Service Agent (ESA) またはハードウェア管理コンソール (HMC) を使用して IBM に送信するには、「**Enable PM Data Transmission**」を使用します。

- **Disable PM Data Transmission**

  IBM へのパフォーマンス・データの送信を使用不可にするには、「**Disable PM Data Transmission**」を使用します。

- **Retransmit Recorded Data**

  以前に記録されたパフォーマンス・データを再送するには、「**Retransmit Recorded Data**」を使用します。

  ```
  Retransmit Recorded Data

  Type or select values in entry fields.
  Press Enter AFTER making all desired changes.

                                                [Entry Fields]
  * Enter the Date [YYYYMMDD]                   []                      #
  ```

  – 2009 年 2 月 12 日の PM データを再送するには、テキスト・ボックスに `20090212` を入力します。
  – 記録された使用可能な PM データ・ファイルをすべて送信するには、`0` を入力します。

- 日付を入力すると、ESA または HMC を使用して IBM にデータを送信するための次の手順が表示されます。

```
Steps to do a Manual transmission from Electronic Service Agent on the HMC

1.  Login to HMC
2.  Select 'Service Management'
3.  Select 'Transmit Service Information'
4.  Click the 'Send' button labeled 'To transmit the performance management information immediately,
    click Send.  (the second Send button on the page)
5.  Check the Console Events log for results

Steps to do a Manual transmission from Electronic Service Agent for AIX

1. Login to ESA using web interface (https://hostname:port/esa)
2. Go to 'Service information'
3. Select action 'Collect information'
4. Select the Performance Management checkbox
5. Click OK
6. Check the Activity Log for results"
```

- **Change/Show Customer Information**

  お客様の情報を表示または更新するには、「**Change/Show Customer Information**」を使用します。 PM データの送信が使用可能になっている場合は、お客様の情報が IBM に送信されます。

- **Change/Show Data Retention Period**

  データ保存期間 (Data Retention Period) を表示または変更するには、「**Change/Show Data Retention Period**」を使用します。 保存期間は、既存のデータが削除されるまでにデータ・ディレクトリーに保持される期間を示します。

- **Change/Show Trending Days and Shift Timing**

  傾向日数 (Trending Days) とシフト・タイミング (Shift Timing) を表示または更新するには、「**Change/Show Trending Days and Shift Timing**」を使用します。

ワークロード評価プログラムのセットアップ**:**

このメニューは、ワークロード評価プログラムのセットアップと構成に使用されます。

```
  WLE

Type or select values in the entry fields.
Press Enter AFTER making all desired changes.

  WLE Collection
  WLE input type
```

- **WLE Collection**

  WLE Collection を使用して、WLE への入力として使用するレポート作成を使用可能または使用不可にします。 このフィールドを使用して、**WLE Collection** の現在の状態をデフォルトで提供します。 関連する記録を停止できるようにするには、コレクションを使用不可にする必要があります。

- **WLE input type**

  WLE input type を使用して、WLE レポートが現在実行中のローカル・バイナリー記録に基づくか、またはローカルの **nmon** 記録に基づくかどうかを決定します。 このことが適用できるのは、永続記録の場合に限定されることに注意してください。

パフォーマンス・マネージメント (PM) サービスを使用した連続システム・パフォーマンス・モニター

パフォーマンス・マネージメント (PM) サービスにより、システム・パフォーマンス・データの収集、アーカイブ、および分析が自動化され、お客様がシステムのリソースと容量を管理するのに役立つレポートが戻されます。 収集されるデータは、システム使用状況、パフォーマンス情報、およびハードウェア構成情報です。

収集されたパフォーマンス・マネージメント (PM) データは、Electronic Service Agent (ESA) またはハードウェア管理コンソール (HMC) を使用して IBM に送られます。 IBM はお客様のデータを保管し、サーバーの拡張とパフォーマンスを示す一連のレポートとグラフを提供します。 お客様はこれらのレポートに従来のブラウザーを使用して電子的にアクセスできます。

この機能を IBM Systems Workload Estimator と一緒に使用すると、お客様のビジネス・トレンドが中央処理装置 (CPU) やディスクのような必須ハードウェアのアップグレードのタイミングとどのように関連しているかを明確に理解できます。 IBM Systems Workload Estimator はシステムの統合をサイジングし、ロジカル・パーティションを使用したシステムのアップグレードを見積もることができます。この場合、PM for IBM Power Systems™ で複数のシステムまたはパーティションのデータを IBM Systems Workload Estimator に送信します。

パフォーマンス・マネージメント・サービスでは、**topasrec** 永続バイナリー記録を使用してパフォーマンス・データを収集します。 このため、**topasrec** 永続バイナリー記録は、PM サービスでパフォーマンス・データを収集するために常に使用可能にしておく必要があります。

注**:**

1. お客様は PM サービスを使用することにより、IBM が PM for IBM Power Systems サーバーで収集されたデータを IBM 社内で無制限に共有して、問題判別、お客様のパフォーマンスおよびキャパシティー計画立案支援、お客様と IBM の既存および新規ビジネス関係の維持、お客様への既存または予測されるリソース制約の通知、IBM 製品を強化するため使用することに同意されるものとします。 また、ユーザー・データは EU の加盟国であるかどうかにかかわらず、任意の国の法人または個人 (IBM、IBM ビジネス・パートナーまたは IBM と契約した第三者) に転送されることがあることにも同意するものとします。
2. お客様は、IBM がお客様のデータを第三者 (1 つ以上のソリューション・プロバイダーおよび IBM ビジネス・パートナーを含む) と共有して、お客様のパフォーマンスとキャパシティー需要を認識し、よりレベルの高いサービスをお客様に提供できるように IBM に許可を与えることができます。この許可は、お客様がオンラインでグラフを表示したときに出されます。

パフォーマンス・マネージメント・サービスの詳細については、`README.perf.tools` ファイルを参照してください。

初期パフォーマンスの診断

報告されたパフォーマンス上の問題には、パフォーマンス上の問題を診断するときに考慮すべき多くのタイプがあります。

報告されたパフォーマンス上の問題のタイプ

パフォーマンス上の問題が報告された場合、考えられる原因のリストを絞り込むことによって、そのパフォーマンス上の問題のタイプを判別できると便利です。

特定のプログラムの実行が遅い

プログラムは幾つかの理由のうちの 1 つで、実行が遅くなり始めることがあります。

この状態は重要でないように見えるかもしれませんが、それでも検討すべき問題があります。

- そのプログラムの実行は常に遅いか。

  プログラムの実行が遅くなり始めたばかりなら、最近行った変更がその原因である可能性があります。
- ソース・コードが変更されているか、または新しいバージョンがインストールされているか。

  もしそうであれば、プログラマーまたはベンダーに問い合わせてください。
- 環境に何らかの変更が行われたか。

  プログラムが使用するファイル (それ自体の実行可能プログラムを含む) が移動されている場合、そのプログラムでは、以前は存在しなかったネットワークによる遅延が発生している可能性があります。 あるいは、ファイルが以前は別のディスク上にあったディスク・アクセス機構の 1 つに競合が発生している可能性があります。

  システム管理者がシステム・チューニング・パラメーターを変更した場合は、プログラムが以前には経験しなかった制約を受ける場合があります。 例えば、管理者が優先順位の計算方法を変更すると、フォアグラウンド・プログラムの速度が上がる一方で、バックグラウンドでかなり高速に実行されていたプログラムがスローダウンすることがあります。
- プログラムは、**perl**、**awk**、**csh**、またはその他のインタープリター言語で書かれているか。

  残念ながら、インタープリター言語はコンパイラーによって最適化されません。 また、**perl** または **awk** のような言語を使用すれば、極端に計算集中または入出力集中の操作を数文字で簡単に要求することができます。そのようなプログラムでは一般に、机上検査または非公式な同僚による評価を、各演算に含まれる反復の数に重点を置いて行うと、時間を費やす価値があります。
- プログラムは常に同じ速度で実行されるか、あるいは速くなることがあるか。

  ファイルシステムは、将来参照するファイルのページを保持するために、システム・メモリーの一部を使用します。 ディスク制約のプログラムが 2 回、間断なく実行された場合、通常は 2 回目の方が 1 回目より高速に実行されます。 同様の動作が、NFS を使用するプログラムで見られることもあります。 これは、コンパイラーのような大規模なプログラムで起こることもあります。 プログラムのアルゴリズムはディスク制約ではないかもしれませんが、大規模な実行可能プログラムをロードするのに時間がかかるために、プログラムの最初の実行に要する時間がその後の実行に要する時間より長くなる可能性があります。
- プログラムの実行が常に遅い場合、またはその環境に明らかな変更が行われていないのにスローダウンしている場合は、そのリソースへの依存性を調べてください。

  『パフォーマンス制約リソースの識別』に、ボトルネックを検出する手法についての説明があります。

1 日の特定の時刻にすべての実行が遅くなる

システムが 1 日のうちのある時間にスローダウンする幾つかの理由があります。

ほとんどの人が、ピーク時間のスローダウンを経験しています。これは、組織内の多数の人が習慣的に、毎日 1 回以上の特定時刻にシステムを使用するために起こるものです。 この現象は、必ずしも単にロードの集中のせいとは限りません。 時には、これは負荷が重くなった場合にのみ問題になる不均衡を示すものです。 システム内でその他のソースに繰り返し起こる状態を考慮する必要があります。

- **iostat** および **netstat** コマンドをスローダウンの時間中ずっと実行しているか、直前にモニター・メカニズムからデータをキャプチャーした場合、あるディスクがその他のディスクよりずっと多く使用されているか。 CPU アイドルのパーセンテージが一貫してほぼゼロか。 送受信されるパケットの数が異常に多いか。
  - ディスクの平衡が取れていない場合は、 193 ページの『論理ボリュームおよびディスク入出力のパフォーマンス』を参照してください。
  - CPU が飽和状態の場合は、**ps** または **topas** コマンドを使用して、この期間中に実行されているプログラムを識別してください。 15 ページの『コマンドを使用した連続システム・パフォーマンス・モニター』に記載されているサンプル・スクリプトを使用すると、最も CPU 使用量の多いユーザーの検索が簡単になります。
  - 昼食時の停滞のように、スローダウンの原因がよく理解できない場合は、グラフィック **xlock** やゲーム・プログラムなどの異常なプログラムを探してください。 **xlock** プログラムの一部のバージョンは、使用されていないディスプレイにグラフィック・パターンを表示するために膨大な量の CPU 時間を使用することで知られています。また、誰かが「CPU burner」として知られるプログラムを、最も邪魔にならない時間に実行しようとしている可能性もあります。
- `/var/adm/cron/cron.allow` ファイルが null でない限り、不経済な操作がないか、`/var/adm/cron/crontab` ディレクトリーの内容を調べる必要があります。

問題の根幹がフォアグラウンド・アクティビティーと長時間実行される、CPU 集中プログラム (バックグラウンドで実行される、または実行すべきプログラム) との競合から生じていることが判明した場合は、**schedo** コマンドを使用して、フォアグラウンドの方に高い優先順位を与えるように優先順位の計算方法を変更することを考慮してください。 135 ページの『スレッド優先順位の値の計算』を参照してください。

予期しない時刻にすべての実行が遅くなる

この状態に対する最善のツールは、**filtd** デーモン (PTX のコンポーネント) などの過負荷検出機能です。

filtd デーモンをセットアップすれば、特定の条件が検出された場合にシェル・スクリプトを実行したり、特定の情報を収集したりすることができます。 **vmstat**、**iostat**、**netstat**、**sar**、および **ps** コマンドを含むシェル・スクリプトを使用して、より特化したメカニズムを構成することができます。

分散環境内の 1 つのシステムでのみ問題が発生する場合は、異常なプログラムが作動しているか、または 2 つのプログラムがランダムに交差していることが考えられます。

特定のユーザーのすべての実行が遅い

システムが特定の個人に影響しているように思えることがあります。

- このケースの解決方法は、問題を定量化することです。 ユーザーにどのコマンドを頻繁に使用するか尋ね、次の例のように、**time** コマンドを指定してそれらのコマンドを実行してください。

  ```
  # time cp .profile testjunk
  real    0m0.08s
  user    0m0.00s
  sys     0m0.01s
  ```

  その後、同一のコマンドをパフォーマンス問題を経験していないユーザー ID で実行します。 報告された `real` の時間に差はありますか。
- プログラムは、実行ごとの CPU 時間 (`user+sys`) に大きな差は示さないはずですが、入出力が多いかより低速なために、リアルタイムには差が出ます。ユーザーのファイルは NFS マウント・ディレクトリーにありますか。 または、その他の理由で高活動のディスク上にありますか。

- ユーザーの `.profile` ファイルに異常な *$PATH* 指定がないか検査してください。例えば、`/usr/bin` を検索する前に、必ず 2、3 の NFS マウント・ディレクトリーを検索すると、すべての処理により時間がかかります。

多数の LAN 接続システムが同時にスローダウンする

独立システムから分散システムへの移行の際に、幾つかの共通の問題が生じます。

問題は通常、できるだけ早く新しい構成を稼働させる必要から、あるいは特定の機能のコストについての認識が欠如しているために起こります。最大伝送単位 (MTU) および mbufs に関する LAN 構成のチューニングに加えて、その時点では妥当と思われた一連の決定の結果生じた、LAN 特有の異常または最適ではない状態を探してください。

- ネットワーク統計情報を使用して、ネットワークに物理的な問題がないことを確認します。 **netstat -v**、**entstat**、**tokstat**、**atmstat**、または **fddistat** などのコマンドで、アダプターに過度のエラーまたは衝突が示されていないことを確認してください。
- ソフトウェアまたはファームウェアのバグの中には、ブロードキャストまたはその他のパケットで、LAN を散発的に飽和状態にさせてしまうものがあります。

  ブロードキャスト・ストームが発生すると、ネットワークをアクティブに使用していないシステムでさえ、絶え間のない割り込みや、パケットの受信および処理に CPU リソースが消費されることにより、スローダウンすることがあります。 これらの問題の検出や特定には、通常のパフォーマンス・ツールよりも、LAN 分析装置の方が適しています。
- 2 つの LAN を 1 つのシステムを介して接続していますか。

  システムをルーターとして使用すると、パケットを処理し、コピーするために大量の CPU 時間を消費します。 また、システムによって処理される他の作業からの干渉を受けることもあります。専用ハードウェア・ルーターおよびブリッジは通常、費用対効果がより高く、堅固なソリューションです。
- それぞれの NFS マウントごとに、明らかな目的がありますか。

  分散構成の開発のある段階で、NFS マウントは、新しいシステムのユーザーに、元のシステムのホーム・ディレクトリーへのアクセスを与えるために使用されます。このような使い方をすれば初期の移行は単純になりますが、データ通信コストの負担が続くことになります。 システム A のユーザーに主としてシステム B のデータを、システム B のユーザーに主としてシステム A のデータを使わせているというようなことは、知られていないことではありません。

  NFS を介してファイルにアクセスすると、LAN トラフィック、クライアントおよびサーバーの CPU 時間、およびエンド・ユーザーの応答時間にかなりのコストがかかることになります。 一般的なガイドラインは、ユーザーとデータは通常同じシステム上にあるべきだということです。 例外は、オーバーライドの問題に、リモートのデータに余分の費用と時間をかけるだけの価値があると判断できる場合です。幾つかの例として、より信頼性のあるバックアップおよび制御のためにデータを集中させる必要がある場合、または、すべてのユーザーが確実にプログラムの最新バージョンを使用する必要がある場合が挙げられます。

  これらの必要性やその他の必要性によって、NFS クライアント/サーバー交換の有効なレベルが要求される場合は、多数のシステムを、一部はサーバー、一部はクライアントとして持つより、1 つのシステムをサーバーの役割専用にする方が適しています。
- プログラムはリモート・プロシージャー・コール (RPC) を使用するため、正確かつ正当に移植されていますか。

プログラムを分散環境に移植する最も簡単な方法は、プログラム呼び出しを 1:1 で RPC に置き換えることです。 あいにく、ローカル・プログラム呼び出しと RPC のパフォーマンスの格差は、ローカル・ディスク入出力と NFS 入出力との格差よりさらに大きくなります。RPC が本当に必要であるという前提の場合は、可能であれば必ず、それらをバッチ処理するようにしてください。

特定のサービスやデバイスですべてが時々スローダウンする

特定のサービスやデバイスで、すべてが時々スローダウンするさまざまな理由があります。

特定のサービスまたはデバイスを使用するものすべてが時々スローダウンする場合は、その特定のサービスまたはデバイスが説明されている章を参照してください。

- 112 ページの『マイクロプロセッサーのパフォーマンス』
- 138 ページの『メモリー・パフォーマンス』
- 193 ページの『論理ボリュームおよびディスク入出力のパフォーマンス』
- 254 ページの『ファイルシステムのパフォーマンス』
- 322 ページの『ネットワーク・パフォーマンスの分析』
- 362 ページの『NFS パフォーマンスのモニターおよびチューニング』

リモート接続時にすべての実行が遅くなる

システムへのローカルおよびリモート認証の動作は非常に異なる場合があります。デフォルトでは、ユーザーがユーザー ID を使ってログインする際に、まずローカル認証ファイルが使用されます。 これは、ネットワーク・ベースの認証機構と比較してより高速です。

ユーザーがなんらかのネットワーク認証機構を使用してログインと認証を行う場合、これがユーザー ID のルックアップ時に最初に検索される機構となります。 これは、ユーザーのログイン名のルックアップを行うコマンドに影響します。また、以下のコマンドにも影響します。

- **ps -ef**
- **ls -l**
- **ipcs -a**

/usr/lib/security/methods.cfg ファイルには、特定の認証プログラムが定義されています。デフォルト値は compat で、これはローカル認証方式です。 特定のユーザー ID の現在の認証設定を参照するには、そのユーザー ID を使ってログインし、コマンド・ラインで次のように入力します。

```
# echo $AUTHSTATE
```

まず、ローカル認証機構を最初に使い、次に DCE のようなネットワーク・ベースの認証を使用したい場合には、次のコマンド・ラインを入力します。

```
# export AUTHSTATE="compat,DCE"
```

パフォーマンス制約リソースの識別

マルチユーザー・ワークロードの実行中にリソースの使用率を全体的に調べるための最善のツールは、**vmstat** コマンドです。

vmstat コマンドは、メモリー使用率データに加えて、CPU およびディスク入出力アクティビティーも報告します。 以下の **vmstat** コマンドをインスタンス化すると、5 秒ごとにシステム・アクティビティーについて 1 行の要約レポートが生成されます。

```
# vmstat 5
```

上記の例では、間隔の後に回数が指定されていないため、コマンドをキャンセルするまでレポート生成は継続されます。

AIXwindows および幾つかの合成アプリケーションを実行中のシステムについて、以下の **vmstat** レポートが作成されました (例としてわかりやすくするために、一部の低アクティビティーの間隔は削除されています)。

```
kthr     memory             page              faults        cpu
----- ----------- ------------------------ ------------ -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 0  0  8793    81   0   0   0   1    7   0 125   42  30  1  2 95  2
 0  0  8793    80   0   0   0   0    0   0 155  113  79 14  8 78  0
 0  0  8793    57   0   3   0   0    0   0 178   28  69  1 12 81  6
 0  0  9192    66   0   0  16  81  167   0 151   32  34  1  6 77 16
 0  0  9193    65   0   0   0   0    0   0 117   29  26  1  3 96  0
 0  0  9193    65   0   0   0   0    0   0 120   30  31  1  3 95  0
 0  0  9693    69   0   0  53 100  216   0 168   27  57  1  4 63 33
 0  0  9693    69   0   0   0   0    0   0 134   96  60 12  4 84  0
 0  0 10193    57   0   0   0   0    0   0 124   29  32  1  3 94  2
 0  0 11194    64   0   0  38 201 1080   0 168   29  57  2  8 62 29
 0  0 11194    63   0   0   0   0    0   0 141  111  65 12  7 81  0
 0  0  5480   755   3   1   0   0    0   0 154  107  71 13  8 78  2
 0  0  5467  5747   0   3   0   0    0   0 167   39  68  1 16 79  5
 0  1  4797  5821   0  21   0   0    0   0 191  192 125 20  5 42 33
 0  1  3778  6119   0  24   0   0    0   0 188  170  98  5  8 41 46
 0  0  3751  6139   0   0   0   0    0   0 145   24  54  1 10 89  0
```

この最初の評価では、page カテゴリーの *pi* および *po* 列と、cpu カテゴリーの 4 列に特に注目してください。

pi および *po* エントリーは、ページング・スペースのページインとページアウトをそれぞれ表します。 ページング・スペース入出力のインスタンスを監視すると、ワークロードがシステム・メモリー制限に近づいているか超えている場合があります。

us および *sy* (ユーザーおよびシステム) の CPU 使用率のパーセンテージの合計が、所定の 5 秒間に 90% より大きい場合、ワークロードは、その間隔中にシステムの CPU 制限に近づいています。

入出力待ち *wa* のパーセンテージがゼロに近い場合、かつ *pi* および *po* の値がゼロの場合は、システムは重複していないファイル入出力の待ちに時間を消費し、ワークロードの一部が入出力制約の状態になります。

vmstat コマンドによってかなりの量の入出力待ち時間が示された場合は、**iostat** コマンドを使用してさらに詳細な情報を収集してください。

以下のように **iostat** コマンドをインスタンスすると、5 秒ごとに入出力アクティビティーおよび CPU 使用率について 1 行の要約レポートが生成されます。さらに、間隔の後にカウントとして 3 を指定しているので、3 つ目のレポートの後、レポートは終了します。

```
# iostat 5 3
```

上記の **vmstat** サンプル内のものと同じワークロードを、異なる時刻に実行するシステムについて、以下の **iostat** レポートが作成されました。 最初のレポートは、それに先立つブート以後の累積アクティビティーを表し、その後のレポートは、それより前の 5 秒間隔のアクティビティーを表しています。

```
tty:      tin         tout   avg-cpu:  % user    % sys     % idle    %iowait
          0.0          4.3               0.2       0.6       98.8       0.4

Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk0           0.0       0.2       0.0       7993      4408
```

```
hdisk1          0.0       0.0       0.0       2179      1692
hdisk2          0.4       1.5       0.3      67548     59151
cd0             0.0       0.0       0.0          0         0

tty:      tin         tout   avg-cpu:  % user    % sys     % idle    %iowait
          0.0         30.3                8.8      7.2       83.9       0.2

Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk0           0.2       0.8       0.2          4         0
hdisk1           0.0       0.0       0.0          0         0
hdisk2           0.0       0.0       0.0          0         0
cd0              0.0       0.0       0.0          0         0

tty:      tin         tout   avg-cpu:  % user    % sys     % idle    %iowait
          0.0          8.4                0.2      5.8        0.0      93.8

Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk0           0.0       0.0       0.0          0         0
hdisk1           0.0       0.0       0.0          0         0
hdisk2          98.4     575.6      61.9        396      2488
cd0              0.0       0.0       0.0          0         0
```

最初のレポートは、このシステムの入出力が不均衡状態であることを示しています。 入出力の大部分 (読み取り K バイト数の 86.9% および書き込み K バイト数の 90.7%) は、hdisk2 に対するもので、これにはオペレーティング・システムとページング・スペースの両方が含まれます。 ブート以降の *CPU* 使用率累積統計値は、システムが 1 日 24 時間連続して使用されているのでなければ、通常無意味です。

2 番目のレポートは、hdisk0 から読み取り中の少量のディスク・アクティビティーを示します。これには、システムの 1 次ユーザー用の別個のファイルシステムが含まれます。 CPU アクティビティーは、2 つのアプリケーション・プログラムおよび **iostat** コマンドそのものから生じます。

3 番目のレポートでは、大量のメモリー (上記の例では約 26 MB) を割り当て、そこに保管するプログラムを実行して、人工的にスラッシングに近い条件を作り出していることが分かります。 さらに上記の例では、hdisk2 はその時間の 98.4% がアクティブで、その結果 93.8% の入出力待ちとなります。

単一プログラムの制限要因

読者がシステムのただ一人のユーザーの場合は、**time** コマンドを次のように使用すれば、プログラムが入出力依存か CPU 依存かについての概念を得ることができます。

```
# time cp foo.in foo.out

real    0m0.13s
user    0m0.01s
sys     0m0.02s
```

注**: time** コマンドの例は、どれも Korn シェル **ksh** 内に構築されるバージョンを使用しています。 公式の **time** コマンド /usr/bin/time によるレポートの精度はもっと低くなります。

上記の例では、**cp** プログラムを実行するための**実際の**経過時間 (0.13 秒) が、ユーザーおよびシステム CPU 時間の合計 (0.03 秒) よりかなり大きいという事実が、このプログラムが入出力制約であることを示しています。 これは主として foo.in ファイルが最近読み取られていないために起こることです。

SMP では、出力は新しい意味を表します。 詳しくは、 123 ページの『time および timex コマンドの考慮事項』を参照してください。

同じファイルに対して数秒後に同じコマンドを実行すると、以下の出力が得られます。

```
real    0m0.06s
user    0m0.01s
sys     0m0.03s
```

foo.in ファイルのほとんど、またはすべてのページは、まだメモリー内にあります。それは、それらのページを再利用するプロセスが存在しなかったことと、このファイルがシステム上の RAM の量と比較して小さいことによります。 小さい foo.out ファイルはメモリー内のバッファーにも入れられ、それを入力として使用するプログラムは、ディスク依存性をほとんど示しません。

プログラムのディスク依存性を判別しようとする場合は、その入力が信用できる状態にあることを確認する必要があります。 つまり、プログラムが通常、最近アクセスされていないファイルに対して実行される場合は、そのプログラムの測定に使用されるファイルがメモリー内にないことを確認する必要があります。これに対して、プログラムが通常、標準シーケンス (プログラムがその入力を先行するプログラムの出力から得る) の一部として実行される場合は、測定が信用できるものであることを確認するために、メモリーに情報を与えておく必要があります。 例えば、以下のコマンドには、メモリーを foo.in ファイルのページで満たす効果があります。

```
# cp foo.in /dev/null
```

ファイルが RAM に比較して大きい場合は、状態はもっと複雑です。 あるプログラムの出力がその次のプログラムの入力であり、ファイル全体が RAM に収まらない場合、2 番目のプログラムはそのファイルの先頭にあるページを読み取りますが、それによって、ファイルの終わりのページはずれることになります。この状態を確実にシミュレートするのは非常に困難ですが、ほぼディスク・キャッシングが行われない状態に相当します。

ファイルが RAM よりわずかに大きい場合、これは RAM 対ディスクの分析の特殊なケースであり、次のセクションで説明します。

ディスクまたはメモリー関連の問題

実メモリーのかなりの部分をファイルのバッファリングに使用できるのと同じように、システムのページ・スペースを、RAM から排除されたプログラム作業データの一時記憶域として使用することができます。

データの読み取りをほとんど、またはまったく行わないプログラムがあって、しかもそれが入出力依存の徴候を示しているとします。 さらに悪いことに、ユーザー + システム時間に対するリアルタイムの率は、その後の実行でも改善されません。 このプログラムはおそらくメモリー制約で、その出力はページング・スペースに対して行われ、入力もページング・スペースから行われると思われます。 この可能性について調べる方法は、次の **vmstatit** シェル・スクリプトに示されています。

```
vmstat -s >temp.file   # cumulative counts before the command
time $1                # command under test
vmstat -s >>temp.file  # cumulative counts after execution
grep "pagi.*ins" temp.file >>results   # extract only the data
grep "pagi.*outs" temp.file >>results  # of interest
```

vmstatit スクリプトは、システムの始動以後の多数のシステム・アクティビティーの累積数を示す、おびただしい量の **vmstat -s** レポートを要約します。

シェル・スクリプトが次のように実行されると、

```
# vmstatit "cp file1 file2"  2>results
```

その結果は次のとおりです。

```
real    0m0.03s
user    0m0.01s
sys     0m0.02s
```

```
     2323 paging space page ins
     2323 paging space page ins
     4850 paging space page outs
     4850 paging space page outs
```

ページング前とページング後の統計情報は同一で、これは、**cp** コマンドはページング制約ではないという確信を強めるものです。 **vmstatit** シェル・スクリプトの拡張変種を使用して、次のように、真の状態を表示することができます。

```
vmstat -s >temp.file
time $1
vmstat -s >>temp.file
echo "Ordinary Input:"               >>results
grep "^[ 0-9]*page ins"    temp.file >>results
echo "Ordinary Output:"              >>results
grep "^[ 0-9]*page outs"   temp.file >>results
echo "True Paging Output:"           >>results
grep "pagi.*outs"          temp.file >>results
echo "True Paging Input:"            >>results
grep "pagi.*ins"           temp.file >>results
```

オペレーティング・システムのファイル入出力は VMM によって処理されるので、**vmstat -s** コマンドは通常のプログラムの入出力をページインおよびページアウトとして報告します。 **vmstatit** シェル・スクリプトの以前のバージョンが、最近読み取られていないラージ・ファイルの **cp** コマンドに対して実行された場合、その結果は次のようになりました。

```
real    0m2.09s
user    0m0.03s
sys     0m0.74s
Ordinary Input:
    46416 page ins
    47132 page ins
Ordinary Output:
   146483 page outs
   147012 page outs
True Paging Output:
     4854 paging space page outs
     4854 paging space page outs
True Paging Input:
     2527 paging space page ins
     2527 paging space page ins
```

time コマンド出力で、入出力依存性があることが確認できます。 ページインの増加は、**cp** コマンドの要求を満たすために必要な入出力を示します。 ページアウトの増加は、ファイルのサイズが、メモリーからのダーティー・ページ (必ずしもそのファイル自体のものではない) の作成を強制するのに十分な大きさであることを示します。 累積ページング・スペース入出力の数に変更がないという事実から、**cp** コマンドが、テスト・マシンのメモリーを過負荷にするだけの大きさのデータ構造体は構築していないことが確認できます。

このバージョンの **vmstatit** スクリプトが入出力を報告する順序には意図があります。 通常のプログラムでは、ファイル入力を読み取ってから、ファイル出力を書き込みます。 それに対して、ページング・アクティビティーでは通常、収まりきらない作業セグメント・ページの書き出しから始まります。 ページの読み込みは、プログラムがそのページへのアクセスを試みた場合にのみ行われます。 テスト・システムが、ブートされて以後、「`paging space page ins`」のほぼ 2 倍の「`paging space page outs`」があったという事実は、このシステムで実行されているプログラムの少なくとも一部は、プログラムの終了前に再度アクセスされなかったデータをメモリー内に保管していることを示しています。 103 ページの『メモリー制約プログラム』は、さらに多くの情報を提供します。 138 ページの『メモリー・パフォーマンス』も参照してください。

上記の統計情報におけるメモリー制限の影響を示すために、以下の例では、十分なメモリー (32 MB) のある環境で所定のコマンドを監視し、次に **rmss** コマンドを使用して人為的にシステムを縮小しています (154 ページの『rmss コマンドによるメモリー所要量の評価』を参照してください)。 以下のコマンド・シーケンス

```
# cc -c ed.c
# vmstatit "cc -c ed.c" 2>results
```

は、まずメモリーを C コンパイラーの 7944 行のソース・ファイルおよび実行可能ファイルで満たし、次に 2 番目の実行の入出力アクティビティーを測定します。

```
real    0m7.76s
user    0m7.44s
sys     0m0.15s
Ordinary Input:
    57192 page ins
    57192 page ins
Ordinary Output:
   165516 page outs
   165553 page outs
True Paging Output:
    10846 paging space page outs
    10846 paging space page outs
True Paging Input:
     6409 paging space page ins
     6409 paging space page ins
```

明らかに、これは入出力制約のではありません。 ソース・コードを読み取るための入出力さえ不要です。その後、次のコマンド、

```
# rmss -c 8
```

を実行してマシンの実効サイズを 8 MB に変更し、同じコマンド・シーケンスを実行すると、以下の出力が得られます。

```
real    0m9.87s
user    0m7.70s
sys     0m0.18s
Ordinary Input:
    57625 page ins
    57809 page ins
Ordinary Output:
   165811 page outs
   165882 page outs
True Paging Output:
    11010 paging space page outs
    11061 paging space page outs
True Paging Input:
     6623 paging space page ins
     6701 paging space page ins
```

以下の入出力依存性の徴候が存在します。

- 合計 CPU 時間より長い経過時間
- コマンドの *nth* の実行における相当量の通常入出力

メモリーに制約されない状態より経過時間が長いという事実と、ページング・スペース入出力がかなりの量存在することから、コンパイラーがメモリーの不足により活動を阻害されていることは明らかです。

注: この例は、メモリー制約の影響を示しています。 他のプロセスでメモリーの使用を最小化する努力が払われなかったので、コンパイラーがこの環境でページングを強制された絶対サイズは、意味のある測定値を構成しません。

次の再始動まで、人為的に縮小されたマシンで作業が行われないようにするには、

```
# rmss -r
```

を実行し、**rmss** コマンドが一時差し押さえたメモリーを解放してオペレーティング・システムに戻すことにより、システムをその通常の容量に復元します。

ワークロード・マネージメントの診断

ワークロード・マネージメントとは、単にワークロードのコンポーネントの優先順位を評価することです。

プログラムのパフォーマンス改善とシステム・チューニングの手段をすべて試みたにもかかわらず、パフォーマンスがまだ満足できる水準に達しない場合があります。この場合は、次の 3 つの方法があります。

- 現状のままにしておく
- パフォーマンスの改善を阻害しているリソースをアップグレードする
- ワークロード・マネージメント手法を採用する

最初のアプローチは、一部のユーザーのフラストレーションや生産性の低下につながります。 リソースをアップグレードする場合は、費用の正当性を示せるようでなければなりません。 したがって、明らかな解決策は、ワークロード・マネージメントの可能性を探ることです。

通常は、延期できる作業があります。 例えば、朝一番に必要なレポートを午前 3 時に実行しても、前日の午後 4 時に実行しても、効果は同じです。 違いは、午前 3 時のほうが CPU サイクルおよびその他のリソースがほとんどアイドル中であるということです。特定の時刻または一定間隔でプログラムを実行するように要求する場合は、 **at** または **crontab** コマンドを使用します。

同様に、どうしても日中に実行する必要がある一部のプログラムは、優先順位を下げて実行することができます。 完了まで時間がかかりますが、本当に急ぎの処理との競合を減らすことができます。

その他の手法は、マシン間の作業の移動です。例えば、コンパイルはソース・コードが常駐するマシンで実行します。 この種のワークロードの平衡化の場合は、ネットワークの負荷を減らしてサーバーの CPU 負荷を増やすと、最終的には損失になるので、さらに計画とモニターを行う必要があります。

AIX ワークロード・マネージャー (WLM) は、オペレーティング・システム・カーネルの一部です。 WLM は、スケジューラーと仮想メモリー・マネージャー (VMM) が CPU と物理メモリーのリソースをプロセスに割り当てる方法を、システム管理者がよりよく制御できるように設計されています。 ディスク使用も WLM で制御することができます。 これによって、クラスの異なるジョブが互いに干渉しあうのを防ぎ、それぞれのユーザー・グループの要件に基づいてリソースを割り当てることができます。 詳細については、「*Server Consolidation on RS/6000®*」を参照してください。

リソース管理

AIX は、リソースを管理するためにシステム・パフォーマンスに関して最も効果のあるチューナブル・コンポーネントを提供します。

特定のチューニングの推奨については、以下を参照してください。

- 112 ページの『マイクロプロセッサーのパフォーマンス』
- 138 ページの『メモリー・パフォーマンス』
- 193 ページの『論理ボリュームおよびディスク入出力のパフォーマンス』
- 279 ページの『ネットワーク・パフォーマンス』

- 356 ページの『NFS パフォーマンス』

プロセッサー・スケジューラーのパフォーマンス

プロセッサー・スケジューラーに関しては、パフォーマンス関連で考慮すべき幾つかの問題があります。

スレッド・サポート

スレッド は、低オーバーヘッドのプロセスと考えることができます。 スレッドは、作成するのに必要なリソースがプロセスより少ない、ディスパッチ可能なエンティティーです。 AIX バージョン 4 スケジューラーのディスパッチ可能な基本エンティティーがスレッドです。

プロセスは 1 つ以上のスレッドで構成されています。 事実、以前のリリースのオペレーティング・システムから直接マイグレーションされたワークロードは、プロセスの作成および管理を継続します。 各新規プロセスはその親プロセスの優先順位を持ち、他のプロセスのスレッドとプロセッサーを求めて競合している単一スレッドによって作成されます。 プロセスは実行に使用されるリソースを所有し、スレッドはその現在の状態のみを所有します。

新規または変更されたアプリケーションがオペレーティング・システムのスレッド・サポートを利用して追加のスレッドを作成する場合、それらのスレッドは、プロセスのコンテキスト内に作成されます。 それらのスレッドは、プロセスの専用セグメントおよびその他のリソースを共用します。

プロセス内のユーザー・スレッドには、指定されたコンテンション有効範囲 があります。コンテンション有効範囲がグローバル の場合は、スレッドは、システム内の他のすべてのスレッドと、プロセッサー時間を求めて競合します。 プロセスが作成されるときに作成されるスレッドは、グローバル・コンテンション有効範囲を持っています。 コンテンション有効範囲がローカル の場合、スレッドは、プロセス内の他のスレッドとプロセッサー時間に関して競合します。

次にどのスレッドを実行すべきかを決定するためのアルゴリズムを、スケジューリング・ポリシー といいます。

プロセスとスレッド

プロセスとは、コマンド、シェル・プログラム、または別のプロセスによって始動されるシステム内のアクティビティーです。

プロセスの属性は、次のとおりです。

- pid
- pgid
- uid
- gid
- 環境
- cwd
- ファイル・ディスクリプター
- シグナル・アクション
- プロセス統計情報
- nice

これらの属性は、`/usr/include/sys/proc.h` ファイルに定義されています。

スレッドの属性は、次のとおりです。

- スタック
- スケジューリング・ポリシー
- スケジューリング優先順位
- 保留シグナル
- ブロック化シグナル
- スレッド固有データ

これらのスレッド属性は、/usr/include/sys/thread.h ファイルに定義されています。

各プロセスは、1 つ以上のスレッドから作成されます。 スレッドは、単一の順次制御のフローです。 マルチスレッドの制御によって、アプリケーションは、端末からの読み取りやファイルへの書き込みなどの操作をオーバーラップさせることができます。

さらに、制御のマルチスレッド化によって、アプリケーションは複数ユーザーからのサービス要求に同時に応じることができます。 スレッドは、**fork()** システム・コールを通じて作成されたプロセスのような、複数プロセスの追加オーバーヘッドなしでこれらの機能を提供します。

AIX では、**f_fork()** と呼ばれる高速の fork ルーチンが導入されました。このルーチンは、**fork()** サブルーチンを呼び出した直後に **exec()** サブルーチンを呼び出す、マルチスレッド・アプリケーションの場合に役立ちます。 **fork()** サブルーチンの方が速度は遅くなります。これは、実際に fork する前に fork ハンドラーを呼び出してライブラリー・ロックを獲得し、子に子ハンドラーを実行させて、そのロックを初期化するためです。 **f_fork()** サブルーチンはこれらのハンドラーをバイパスし、 **kfork()** システム・コールを直接呼び出します。 Web サーバーは、**f_fork()** サブルーチンを使用することができるアプリケーションのよい例です。

プロセスとスレッドの優先順位

優先順位管理ツールは、プロセスの優先順位を取り扱います。

AIX バージョン 4 では、プロセス優先順位はスレッド優先順位の先行版にすぎません。 **fork()** サブルーチンが呼び出されると、そのサブルーチンで実行されるプロセスおよびスレッドが作成されます。 スレッドの優先順位は、プロセスの属性であった優先順位です。

カーネルは、それぞれのスレッドごとに優先順位の値 (スケジューリング優先順位 と呼ばれることもある) を維持します。 優先順位の値は正の整数で、関連付けられたスレッドの重要性に逆比例して変化します。つまり、優先順位の値が小さいほど、スレッドの重要性が高いことを意味します。 スケジューラーは、ディスパッチするスレッドを探す際、優先順位の値が最も小さいディスパッチ可能スレッドを選択します。

スレッドの優先順位は、固定優先順位でも非固定優先順位でもかまいません。 固定優先順位スレッドの優先順位の値は定数ですが、非固定優先順位スレッドの優先順位の値は、ユーザー・スレッドの最小優先順位 (定数 40)、スレッドの nice の値 (デフォルトは 20 で、オプションで **nice** または **renice** コマンドによって設定される)、およびそのプロセッサー使用率に伴うペナルティーに基づいて変動します。

スレッドの優先順位は特定の値に固定することができ、その優先順位が **setpri()** サブルーチンを通じて設定 (固定) されている場合、40 より小さい優先順位の値を持つことができます。 これらのスレッドは、スケジューラーの再計算アルゴリズムに影響されません。 その優先順位の値が 40 より小さい値に固定されている場合、これらのスレッドは、どのユーザー・スレッドが実行されるよりも前に実行され、完了します。 例えば、固定値が 10 のスレッドは、固定値が 15 のスレッドより前に実行されます。

ユーザーは **nice** コマンドを適用して、スレッドの非固定優先順位を低くすることができます。 システム管理者はスレッドに負の nice の値を適用して、より高い優先順位を与えることができます。

次の図は、優先順位の値を変更する方法を幾つか示しています。

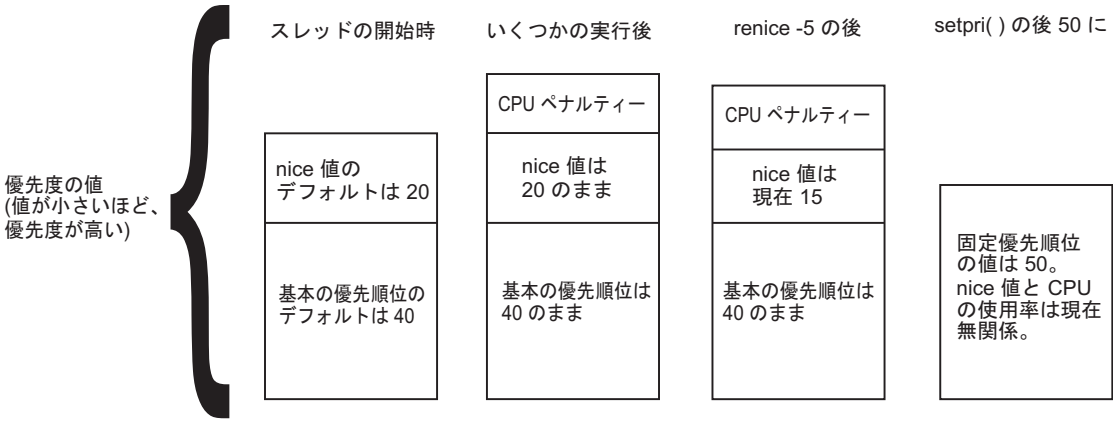


図 *6.* 優先順位の値を判別する方法： この図は、スレッドの実行中や nice コマンドの適用後に、スレッドのスケジューリング優先順位の値がどのように変化するかを示しています。 優先順位の値が小さいほど、スレッド優先順位は高くなります。 開始時は、nice の値はデフォルトの 20 で、基本優先順位はデフォルトの 40 です。いくつかの命令が実行され、プロセッサー・ペナルティーが適用された後も、nice 値は 20 のまま、基本優先順位は 40 のままです。 renice —5 コマンドを実行した後、プロセッサー使用率が前と同じである場合は、nice の値は 15 になり、基本優先順位は 40 のままです。 値 50 を指定して setpri() サブルーチンを実行すると、固定優先順位は 50 になり、nice の値とプロセッサー使用率は無関係になります。

スレッドの nice の値は、スレッドの作成時に設定され、そのスレッドが存在する間一定です。ただし、ユーザーが **renice** コマンドか **setpri()**、**setpriority()**、**thread_setsched()**、または **nice()** システム・コールによって明示的に変更した場合は別です。

プロセッサー・ペナルティーは、スレッドの最新のプロセッサー使用率から計算される整数です。 最新のプロセッサー使用率は、10 ミリ秒のクロックの目盛りの終わりにスレッドがプロセッサーの制御下に入るつど、最大値の 120 に達するまでおよそ 1 ずつ増加します。 ティックごとの実際の優先順位ペナルティーは、**nice** の値と共に増加します。 1 秒に一度、すべてのスレッドの最新のプロセッサー使用率の値は再計算されます。

結果は次のとおりです。

- 非固定優先順位スレッドの優先順位は、その最新のプロセッサー使用率が増加するにつれて下がり、減少するにつれて上がります。 これは、平均して、スレッドが直近で多くのタイム・スライスを割り当てられているほど、そのスレッドに次に割り当てられるタイム・スライスは少なくなる傾向にあるということを示します。
- 非固定優先順位スレッドの優先順位は、その nice の値が増加するにつれて下がり、減少するにつれて上がります。

注**:** 複数プロセッサー実行キューおよびそのロード・バランシング・メカニズムを使用すると、**nice** 値または **renice** 値が、スレッドの優先順位に対して予期した効果を上げないことがあります。これは、優先順位のより低いものが、より高いものと等しい、またはそれ以上の実行時間を与えられることがあるためです。 **nice** または **renice** の予期したとおりの効果を出す必要のあるスレッドは、グローバル実行キューに入れてください。

ps コマンドを使用すれば、プロセスの優先順位の値、nice の値、および短期プロセッサー使用率の値を表示することができます。

nice および **renice** コマンドの使用法の詳細については、 132 ページの『マイクロプロセッサーのコンテンションの制御』を参照してください。

プロセッサー・ペナルティーの計算および最新のプロセッサー使用率の値の減衰についての詳細は、 135 ページの『スレッド優先順位の値の計算』を参照してください。

この優先順位メカニズムは、AIX ワークロード・マネージャーでもプロセッサー・リソース管理を強化するために使用されます。 ワークロード・マネージャー の下でクラス分けされるスレッドの優先順位はワークロード・マネージャーが管理するため、ワークロード・マネージャーの下でクラス分けされていないスレッドとは異なる優先順位を持つ場合があります。

スレッドのスケジューリング・ポリシー

スケジューリング・ポリシーには、スレッドに関する多数の可能な値があります。

SCHED_FIFO
このポリシーを持つスレッドがスケジュールされると、スレッドはブロックされるか、自発的にプロセッサーの制御を引き渡すか、またはより優先順位の高いスレッドがディスパッチ可能にならない限り、完了するまで実行されます。 SCHED_FIFO スケジューリング・ポリシーを持つことができるのは、固定優先順位スレッドだけです。

SCHED_RR
SCHED_RR スレッドは、タイム・スライスの終わりに制御を得ると、その優先順位のディスパッチ可能スレッドのキューの末尾に移動します。 SCHED_RR スケジューリング・ポリシーを持つことができるのは、固定優先順位スレッドだけです。

SCHED_OTHER
このポリシーは、POSIX 標準 1003.4a によりインプリメンテーション定義として定義されます。各クロック割り込みごとに実行中スレッドの優先順位の値を再計算するということは、その優先順位の値が別のディスパッチ可能スレッドの優先順位の値を上回ったために、スレッドが制御を失う場合があることを意味します。

SCHED_FIFO2
このポリシーは SCHED_FIFO の場合と同じですが、ごく短時間スリープしたスレッドが、起動したときにその実行キューの先頭に書き込まれるようにする点が異なります。 この時間枠が親和性制限 (**schedo -o affinity_lim** との調整が可能) です。

SCHED_FIFO3
スケジューリング・ポリシーが SCHED_FIFO3 に設定されているスレッドは、常に実行キューの先頭に書き込まれます。 SCHED_FIFO2 スケジューリング・ポリシーに属するスレッドが SCHED_FIFO3 より先に書き込まれないようにするため、 SCHED_FIFO3 スレッドがエンキューされるときに実行キュー・パラメーターが変更されるので、SCHED_FIFO2 に属するスレッドはどれも、実行キューの先頭に入れる基準を満たすことはできません。

SCHED_FIFO4
優先順位の高い SCHED_FIFO4 スケジューリング・クラスのスレッドは、優先順位の違いが 1 である場合、現在稼働中の優先順位の低いスレッドを優先使用しません。デフォルトの動作では、CPU 上で現在稼働中の優先順位の低いスレッドを、同じプロセッサーで実行資格のある優先順位の高いスレッドが優先使用します。

スケジューリング・ポリシーは **thread_setsched()** システム・コールによって設定され、呼び出し側のスレッドに対してのみ有効です。 ただし、プロセス ID を指定して **setpri()** コールを出すことにより、スレッドを SCHED_RR スケジューリング・ポリシーに設定することもできます。**setpri()** の呼び出し側と **setpri()** のターゲットが一致している必要はありません。

setpri() システム・コールを出すことができるのは、root 権限を持つプロセスだけです。 スケジューリング・ポリシーを任意の SCHED_FIFO オプションまたは SCHED_RR に変更できるのは、root 権限を持つスレッドだけです。 スケジューリング・ポリシーが SCHED_OTHER の場合、優先順位パラメーターは **thread_setsched()** サブルーチンによって無視されます。

スレッドが関係するのは主に、現在幾つかの非同期処理からなっているアプリケーションです。 マルチスレッド構造に変換した場合、これらのアプリケーションがシステムに課するロードは軽くなる可能性があります。

スケジューラー実行キュー

スケジューラーは、ディスパッチ可能になっているすべてのスレッドの実行キューを維持します。

以下の図は、実行キューを象徴化して描いたものです。

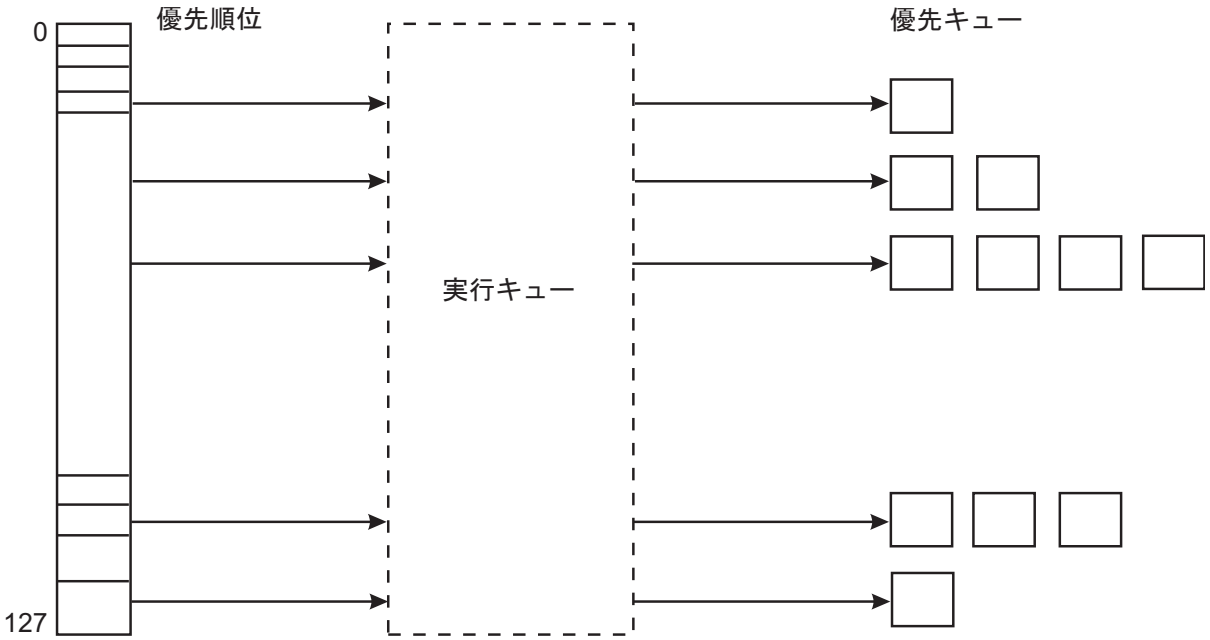


図 7. 実行キュー： この図は、小さい優先順位の値を持つスレッドが、大きい優先順位の値を持つスレッドより先に実行キューを通過することを簡単に示しています。 指定可能な優先順位の値の範囲は 0 から 127 で、この値は合計 128 個の実行キューに直接関連しています。

優先順位を持つディスパッチ可能スレッドはすべて、実行キュー内の位置を占めています。

スケジューラーのディスパッチ可能な基本エンティティーがスレッドです。 AIX は、256 個の実行キューを保持します。これらの実行キューは、各スレッドの優先順位フィールドに入れることが可能な値の範囲 (0 から 255) に直接関連しています。 この方式により、どのスレッドを実行するのが最も好都合かをスケジューラーが判断しやすくなります。 スケジューラーは、単一の大きい実行キューを検索する必要はなく、マスクを使用して、該当する実行キューで実行可能なスレッドの存在を示すビットがオンになっている場所を調べれば済みます。

スレッドの優先順位の値は、迅速かつ頻繁に変化します。一定の移動は、スケジューラーが優先順位を再計算する方法によるものです。ただし、固定優先順位スレッドの場合は、これは正しくありません。

AIX バージョン 6.1 以降では、各プロセッサーはノードごとに実行キューを持ちます。パフォーマンス・ツールに報告される実行キューの値は、各実行キュー内のすべてのスレッドの合計になります。 プロセッサーごとに実行キューを持つと、ロックのディスパッチングにおけるオーバーヘッドが減少し、全体としてのプロセッサーとの親和性が改善されます。 スレッドは、より頻繁に同じプロセッサーにとどまるようになります。 スレッドが、実行可能スレッドが実行されていたプロセッサーとは別のプロセッサー上のイベントのために実行可能になった場合、使用されていないプロセッサーがあると、このスレッドは直ちにディスパッチされます。プロセッサーの状態を調べられるようになるまで (例えば、このスレッドのプロセッサーでの割り込みなど)、優先使用は行われません。

複数の実行キューを持つマルチプロセッサー・システムでは、一時的な優先順位の逆転が起きることがあります。 ある時点で、ある実行キューにあるいくつかのスレッドの優先順位が、他の実行キューのものより高いということがあり得ます。 AIX は、自然に優先順位のバランスをとるメカニズムを備えています。ただし、厳密な優先順位が必要な場合 (例えば、リアルタイム・アプリケーションなど) は、*RT_GRQ* と呼ばれる環境変数が存在します。 *RT_GRQ* 環境変数を ON に設定すると、このスレッドはグローバル実行キューに置かれます。その場合、最高の優先順位を持つスレッドを求めて、グローバル実行キューの検索が行われます。これによって、割り込み駆動型スレッドのパフォーマンスが改善されることがあります。 **schedo** コマンドの *fixed_pri_global* パラメーターが 1 に設定されている場合、固定優先順位で実行中のスレッドはグローバル実行キューに置かれます。

実行キュー内のスレッドの平均数は、**vmstat** コマンド出力の最初の列に表示されます。 この数をプロセッサーの数で除算すると、その結果が各プロセッサーで実行されるスレッドの平均数になります。この値が 1 より大きい場合、これらのスレッドはプロセッサーでの実行の順番を待つ必要があります (数が大きいほど、パフォーマンス遅延が認識されるようになります)。

スレッドが実行キューの終わりに移動されると (例えば、スレッドがタイム・スライスの終わりに制御を得る場合)、そのスレッドは、同じ優先順位の値を持つキュー内の最後のスレッドの後の位置に移動します。

スケジューラー・プロセッサー・タイム・スライス

プロセッサー・タイム・スライスは、スケジューラーが同じ優先順位を持つ別のスレッドに切り替える前に、SCHED_RR スレッドが吸収できる時間の長さです。

schedo コマンドの **timeslice** オプションを使用して、タイム・スライスのクロックの目盛り数を 10 ミリ秒単位で増やすことができます (137 ページの『schedo コマンドによるスケジューラーのタイム・スライスの変更』を参照)。

注**:** タイム・スライスは、保証された量のプロセッサー時間ではありません。 これは、スレッドが別のスレッドに置き換えられる可能性に直面する前に、そのスレッドを制御範囲に置くことができる最長の時間です。 スレッドがフル・タイム・スライスの制御を得る前に、プロセッサーの制御を失う状況はさまざまです。

モード切り替え

ユーザー・プロセスがシステム・リソースへのアクセスを必要とする場合は、そのユーザー・プロセスのモード切り替えが行われます。 これは、システム・コール・インターフェースを通じて、またはページ・フォールトなどの割り込みにより実行されます。

モードには次の 2 つがあります。

- ユーザー・モード
- カーネル・モード

ユーザー・モード (アプリケーションおよび共用ライブラリー) で費やされるプロセッサー時間は、**vmstat**、**iostat**、および **sar** コマンドなどのコマンドの出力に、ユーザー時間として反映されます。 カーネル・モードで費やされるプロセッサー時間は、これらのコマンドの出力に、システム時間として反映されます。

ユーザー・モード:

ユーザー保護ドメインで実行されるプログラムは、ユーザー・プロセスです。

この保護ドメインで実行されるコードはユーザー実行モードで実行され、次のようなアクセスを持っています。

- プロセス専用領域内のユーザー・データに対する読み取り/書き込みアクセス
- ユーザー・テキスト領域および共用テキスト領域への読み取りアクセス
- 共有メモリー機能を使用する共用データ領域へのアクセス

ユーザー保護ドメインで実行されるプログラムは、システム・コールの使用による間接的なアクセスを別にすれば、カーネルまたはカーネル・データ・セグメントにアクセスすることはできません。 この保護ドメイン内のプログラムはそれ自体の実行環境にのみ影響を及ぼすことができ、プロセス状態または非特権状態で実行されます。

カーネル・モード:

カーネル保護ドメインで実行されるプログラムには、割り込みハンドラー、カーネル・プロセス、基本カーネル、およびカーネル・エクステンション (デバイス・ドライバー、システム・コール、およびファイルシステム) が含まれます。

この保護ドメインは、コードがカーネル実行モードで実行されることを示し、次のようなアクセスを持っています。

- グローバル・カーネル・アドレス・スペースへの読み取り/書き込みアクセス
- プロセス内で実行される場合に、プロセス領域のカーネル・データへの読み取り/書き込みアクセス

プロセス・アドレス・スペース内のユーザー・データにアクセスするには、カーネル・サービスを使用する必要があります。

この保護ドメイン内で実行されるプログラムは、次のような特性を持っているため、すべてのプログラムの実行環境に影響を及ぼすことがあります。

- グローバル・システム・データにアクセスできる。
- カーネル・サービスを使用できる。
- セキュリティー上のすべての制限を免れる。
- プロセッサー特権状態で実行される。

モード切り替え:

ユーザー・モード・プロセスでシステム・コールを使用すると、ユーザー・モードからカーネル関数を呼び出すことができます。 直接または間接にシステム・コールを起動する関数へのアクセスは、一般的にはプログラミング・ライブラリーによって提供されます。このライブラリーはオペレーティング・システム関数へのアクセスを提供します。

モード切り替えは、**vmstat** (**cs** 列) および **sar** (*cswch/s*) コマンドの出力に見られるコンテキスト切り替えとは区別する必要があります。 コンテキスト切り替えは、現在実行中のスレッドが、そのプロセッサーで直前に実行されていたスレッドと異なる場合に起こります。

スケジューラーは、以下のいずれかが起こるとコンテキスト切り替えを行います。

- スレッドがディスク入出力、ネットワーク入出力、スリープ、またはロックなどのリソースを (自発的に) 待機しなければならない。
- より優先順位の高いスレッドが (自発的ではなく) ウェイクアップした。
- スレッドがそのタイム・スライス (通常は 10 ミリ秒) を使い果たした。

コンテキスト切り替え時間、システム・コール、デバイス割り込み、NFS 入出力、およびカーネル内のその他のアクティビティーはどれも、システム時間と見なされます。

仮想メモリー・マネージャーのパフォーマンス

仮想アドレス・スペースはセグメントに分割されます。 セグメントは 256 MB の、仮想メモリー・アドレス・スペースの連続した部分で、その中にデータ・オブジェクトをマップすることができます。

プロセスのデータへのアドレス可能度はセグメント (またはオブジェクト) レベルで管理されるので、1 セグメントをプロセス間で共用することも、専用として維持することもできます。例えば、プロセスは、コード・セグメントを共用することができますが、別の専用データ・セグメントを持つこともできます。

実メモリーの管理

VMM は実メモリーの重要な管理ロールを果たします。

仮想メモリー・セグメントは、ページ と呼ばれる固定サイズの単位に分割されます。 POWER5+ プロセッサー上で稼働する AIX 7.1 は、4 種類のページ・サイズ (4 KB、64 KB、16 MB、および 16 GB) をサポートします。詳しくは、複数ページ・サイズ・サポートを参照してください。 セグメント内の各ページは、実メモリー (RAM) に入れることも、必要になるまでディスク上に保管することもできます。 同様に、実メモリーはページ・フレームに分割されます。 VMM の役割は、実メモリーのページ・フレームの割り当てを管理し、現在実メモリーにないか、まだ存在しない仮想メモリー・ページ (例えば、プロセスがそのデータ・セグメントのページに対して初めて参照を行う場合) への、プログラムによる参照を解決することです。

ある瞬間に使用中の仮想メモリーの量が実メモリーより大きくなることがあるので、VMM はディスク上に余りを保管する必要があります。 パフォーマンスの観点から、VMM は 2 つの多少対立する目標を持っています。

- 仮想メモリーを使用する際の、プロセッサー時間およびディスク帯域幅の全般的なコストを最小化する。
- ページ・フォールトの応答時間コストを最小化する。

これらの目標を追求するに当たり、VMM は、ページ・フォールトを満足させるために使用可能なページ・フレームのフリー・リスト を維持します。 VMM は、ページ置換アルゴリズムを使用して、現在メモリー内にあるどの仮想メモリー・ページが、そのページ・フレームをフリー・リストに再割り当てされるかを判別します。 ページ置換アルゴリズムは次のような幾つかのメカニズムを使用します。

- 仮想メモリー・セグメントを永続セグメントまたは作業セグメントのいずれかに分類する。
- 仮想メモリー・セグメントを計算メモリーまたはファイル・メモリーのいずれかを含むものとして分類する。
- そのアクセスがページ・フォールトの原因となる仮想メモリー・ページをトレースする。

- ページ・フォールトを新規ページ・フォールトまたは再ページ・フォールトとして分類する。
- 各仮想メモリー・セグメント内の再ページ・フォールトの比率に関する統計情報を維持する。
- ユーザーによる調整が可能なしきい値が、ページ置換アルゴリズムの決定に影響を及ぼす。

フリー・リスト:

VMM は、ページ・フォールトに対応するために使用する、フリーのページ・フレームの論理リストを維持します。

ほとんどの環境で、VMM は、実行中のプロセスに所有されるページ・フレームの一部を再割り当てすることにより、時々フリー・リストを大きくする必要があります。 再割り当てすべきページ・フレームを持つ仮想メモリー・ページは、VMM のページ置換アルゴリズムによって選択されます。 VMM のしきい値が、再割り当てされるフレームの数を決定します。

永続セグメントと作業セグメント:

永続セグメントは永続的で、一方、作業セグメントは一時的です。

永続セグメントのページは、ディスク上に永続保管場所を持っています。 データまたは実行可能プログラムを含むファイルは、永続セグメントにマップされます。 永続セグメントの各ページは永続ディスク保管場所を持っているので、VMM はページが変更されるとその保管場所にページを書き込み、もう実メモリーにそのページを保持することはできません。 フリー・リスト上に配置するために選択されたとき、ページが変更されていなければ、入出力は必要ありません。 ページが後で再び参照されると、その永続ディスク保管場所から新しいコピーが読み込まれます。

作業セグメントは一時的で、プロセスによって使用される間のみ存在し、永続ディスク保管場所は持っていません。 プロセス・スタックおよびデータ領域は、カーネル・テキスト・セグメント、カーネル・エクステンションのテキスト・セグメント、共用ライブラリーのテキストおよびデータ・セグメントと同様、作業セグメントにマップされます。 作業セグメントのページも、実メモリーに保持されない場合に占めるべき、ディスク保管場所を持っていなければなりません。 ディスク・ページング・スペースは、この目的で使用されます。

次の図は、セグメントのタイプの一部と、そのページのディスク上の場所との関係を示しています。また、ページが実メモリー内にあるときの、そのページの実際の (任意の) 場所も示しています。

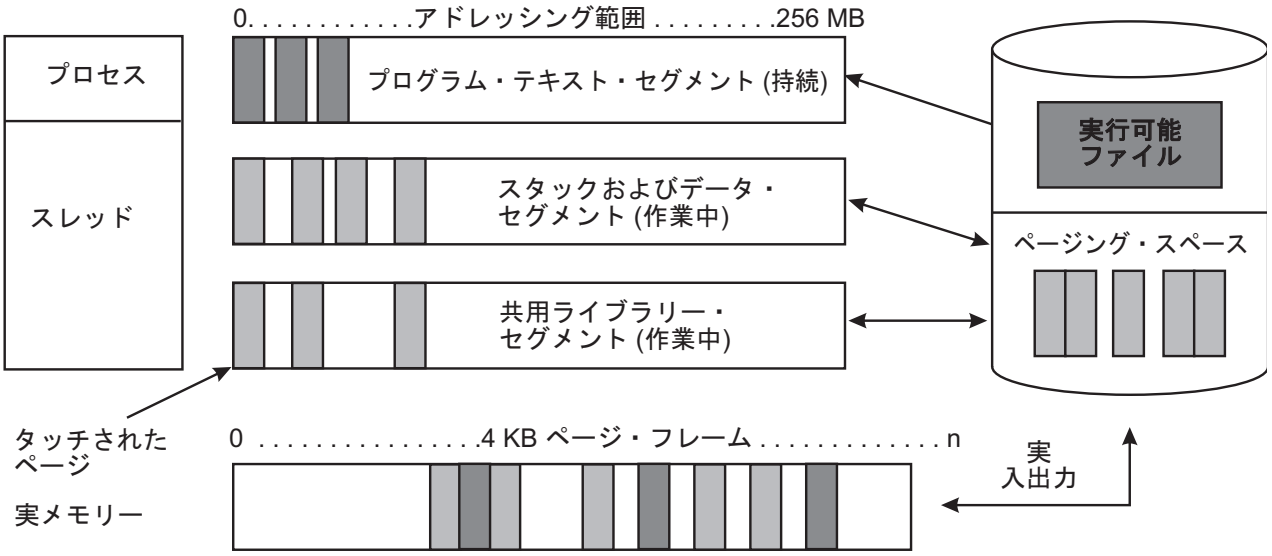


図 *8.* 永続セグメントおよび作業用ストレージ・セグメント： この図は、セグメントのタイプの一部と、そのページのディスク上の場所との関係を示しています。 また、ページが実メモリー内にあるときの、そのページの実際の (任意の) 場所も示しています。 作業セグメントは一時的です。つまり、プロセスによって使用される間のみ存在し、永続ディスク保管場所は持っていません。 プロセス・スタックおよびデータ領域は、カーネル・テキスト・セグメント、カーネル・エクステンションのテキスト・セグメント、共用ライブラリーのテキスト・セグメントおよびデータ・セグメントと同様、作業セグメントにマップされます。 作業セグメントのページも、実メモリーに保持されない場合に占めるべき、ディスク保管場所を持っていなければなりません。 ディスク・ページング・スペースは、この目的で使用されます。

永続セグメントのタイプは、さらに分類されます。 クライアント・セグメント は、リモート実行可能プログラムを含むリモート・ファイル (例えば、NFS を通じてアクセスされるファイル) をマップする際に使用されます。 クライアント・セグメントからのページは、ローカル・ディスクのページング・スペースにではなく、ネットワークを通じて、その永続ファイル・ロケーションに保管され、復元されます。 ジャーナル・セグメントと据え置きセグメント は、アトミックに更新する必要がある永続セグメントです。 ジャーナル・セグメントまたは据え置きセグメントからのページが実メモリーから除去すべきもの (ページアウト) として選択された場合、そのページは、コミットする (その永続ファイル・ロケーションに書き込む) ことが許される状態にない限り、ディスク・ページング・スペースに書き込む必要があります。

計算メモリーとファイル・メモリー**:**

計算メモリー は、計算ページとも呼ばれ、作業用ストレージ・セグメントまたはプログラム・テキスト (実行可能ファイル) セグメントに属するページからなっています。

ファイル・メモリー (またはファイル・ページ) は、残りのページからなっています。 これらは通常、永続ストレージ内の永久データ・ファイルからのページです。

ページ置換**:**

フリー・リスト上の使用可能な実メモリー・フレームの数が少なくなると、ページ・スチール機能が起動されます。 ページ・スチール機能は、ページ・フレーム・テーブル (PFT) 全体を移動して、スチールするページを探します。

PFT には、どのページが参照されたか、およびどのページが変更されたかを知らせるフラグが組み込まれています。 ページ・スチール機能は、参照されたページを検出すると、そのページをスチールせず、代わ

りにそのページの参照フラグをリセットします。 クロック・ハンド (ページ・スチール機能) が次にそのページを渡すときに、参照ビットがまだオフの場合、そのページはスチールされます。 最初のパスで参照されなかったページは、直ちにスチールされます。

変更フラグは、そのページ上のデータが、メモリー内に入れられた後に変更されていることを示します。ページがスチールされることになっていて、変更フラグが設定されている場合は、そのページをスチールする前にページアウト・コールが行われます。 作業セグメントを構成しているページはページング・スペースに書き込まれ、永続セグメントはディスクに書き出されます。

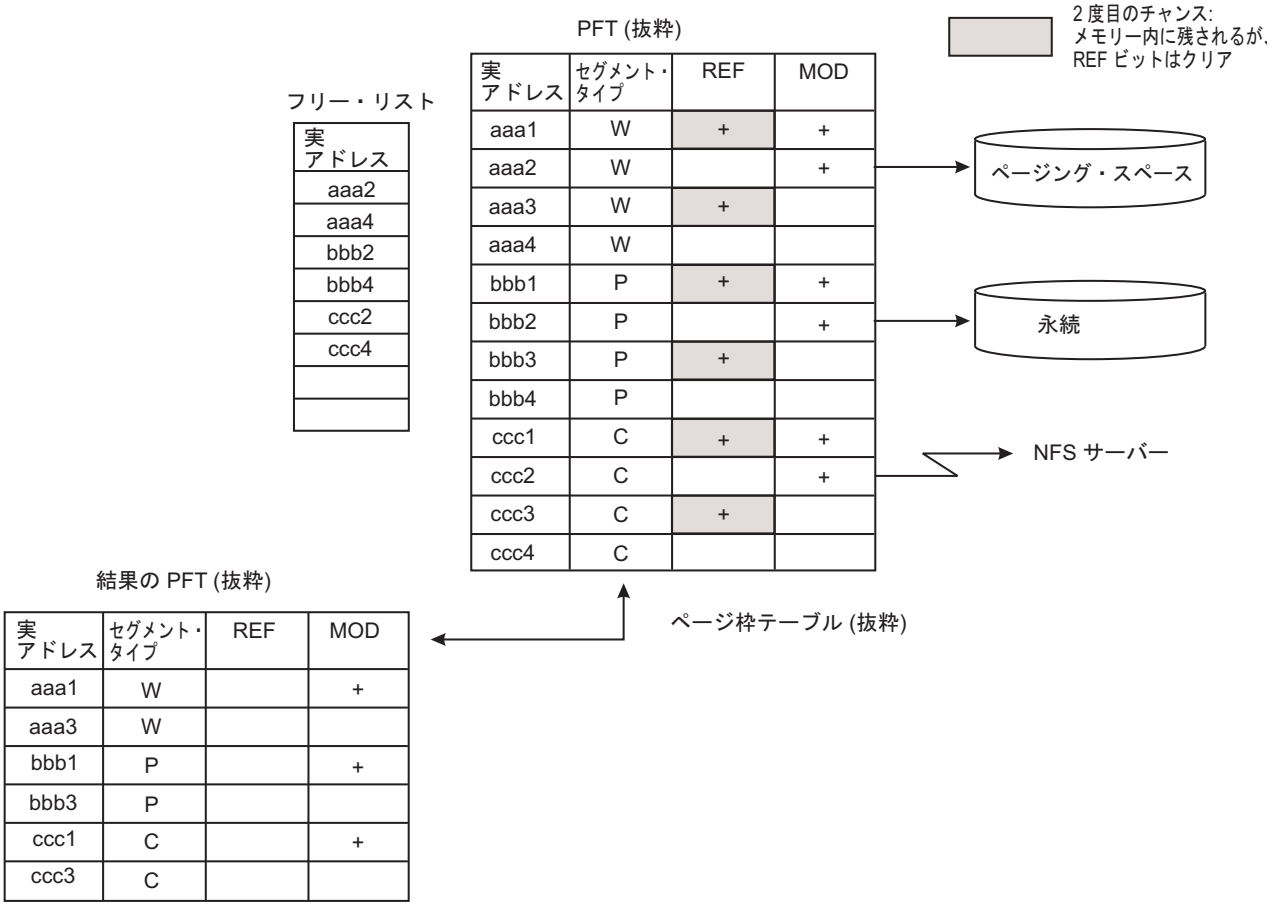


図 *9.* ページ置換の例： この図は、3 つのテーブルを抜粋したものです。 1 つ目のテーブルは、4 つの列があるページ・フレーム・テーブルで、実アドレス、セグメント・タイプ、参照フラグ、および変更フラグを格納しています。 2 つ目のテーブルはフリー・リスト・テーブルと呼ばれ、すべてのフリー・ページのアドレスを格納しています。 最後のテーブルは、すべてのフリー・アドレスが削除された結果のページ・フレーム・テーブルを表しています。

このアルゴリズムは、ページ置換に加えて、新規ページ・フォールト (初めて参照された) および再ページ・フォールト (ページアウトされたページを参照する) の両方を、最新のページ・フォールトの ID を含むヒストリー・バッファーを使用して常時把握しています。 アルゴリズムは次に、ファイル (永続データ) ページアウトと計算 (作業用ストレージまたはプログラム・テキスト) ページアウトのバランスを取ろうとします。

プロセスが終了すると、その作業用ストレージは直ちに解放され、それに関連したメモリー・フレームはフリー・リストに戻されます。 ただし、そのプロセスが開いたファイルはどれも、メモリー内にとどまることができます。

ページ置換は、スレッドがユニプロセッサーで実行中の場合、そのスレッドの有効範囲内で直接行われます。 マルチプロセッサー・システムでは、ページ置換は **lrud** カーネル・プロセスによって行われ、*minfree* しきい値に達したときに CPU にディスパッチされます。 **lrud** カーネル・プロセスは、メモリー・プール当たり 1 スレッドにマルチスレッド化されます。 実メモリーは、CPU の数および RAM の量を基にして、均一サイズのメモリー・プールに分割されます。 システム上のメモリー・プールの数は、**vmstat -v** コマンドを実行して判別できます。

再ページング**:**

ページ・フォールトは、新規ページ・フォールトまたは再ページ・フォールトのいずれかと考えられます。新規ページ・フォールトは、最近参照されたページのレコードが存在しない場合に起こります。 再ページ・フォールトは、最近参照されたことが知られているページが再度参照され、最後にアクセスされてからそのページが置換され (あるいはディスクに書き込まれ) ているために、メモリー内に検出されない場合に起こります。

完ぺきなページ置換ポリシーは、再度参照される予定のないページから常にフレームをスチールすることにより、再ページ・フォールトを完全に除去します (十分な実メモリーがあるという前提で)。 したがって、再ページ・フォールトの数は、頻繁に再利用されるページをメモリー内に保持し、それによって全体としての入出力要求を削減して、システム・パフォーマンスの改善の可能性を探る上で、ページ置換アルゴリズムの有効性を測る逆の尺度です。

ページ・フォールトを新規または再ページに分類するために、VMM は、*N* 個の最新ページ・フォールトのページ ID を入れる、再ページ・ヒストリー・バッファーを維持します。ここで、*N* はメモリーに保持することのできるフレームの数です。 例えば、512 MB のメモリーは 128 KB の再ページ・ヒストリー・バッファーを必要とします。 ページイン時に、ページの ID が再ページ・ヒストリー・バッファー内に検出される場合、それは 1 再ページとしてカウントされます。 また、VMM は、メモリーのタイプごとに再ページ・フォールトの数を維持することによって、計算メモリーの再ページング率とファイル・メモリーの再ページング率を別々に見積もります。 再ページング率には、ページ置換アルゴリズムが実行されるつど 0.9 が乗じられるので、ヒストリー上の再ページング・アクティビティーより強力に最新の再ページング・アクティビティーを反映します。

VMM しきい値**:**

幾つかの数値しきい値が VMM の目標を定義します。これらのしきい値の 1 つが破られると、VMM は適切なアクションをとって、メモリーの状態をバウンダリー内に戻します。 このセクションでは、システム管理者が **vmo** コマンドによって変更できるしきい値について説明します。

フリー・リスト上のページ・フレームの数は、以下のパラメーターによって制御されます。

minfree
: フリー・リスト内の実メモリー・ページ・フレームの許容最小数。フリー・リストのサイズがこの数を下回ると、VMM はページのスチールを開始します。 VMM は、フリー・リストのサイズが *maxfree* に達するまで、ページのスチールを継続します。

maxfree
: VMM のページ・スチールによって増大するフリー・リストの最大サイズ。 フリー・リストのサイズは、プロセスが終了し、その作業セグメント・ページを解放した結果として、またはメモリー内にページを持つファイルを削除した結果として、この数値を超えることがあります。

VMM は、フリー・リストのサイズを *minfree* 以上に保持しようとします。 ページ・フォールトまたはシステム要求が原因でフリー・リストのサイズが *minfree* を下回ると、ページ置換アルゴリズムが実行されます。 幾つかの理由により、フリー・リストのサイズをある一定のレベル (デフォルト値は *minfree*) 以上

に保持する必要があります。例えば、オペレーティング・システムの順次事前取り出しアルゴリズムでは、順次読み取りを行っているプロセスごとに、同時に幾つかのフレームを必要とします。また、VMM はオペレーティング・システムそのものの中のデッドロックも回避する必要があります。これは、ページ・フレームの解放に必要なページを読み込むためのスペースが十分になかった場合に起こることがあります。

以下のしきい値はパーセントで表されます。 これらは、ファイル・ページ (計算セグメント以外のセグメント・ページ) によって占められる、マシンの合計実メモリーの部分を表します。

minperm
ファイル・ページによって占められる実メモリーのパーセンテージがこのレベルを下回ると、ページ置換アルゴリズムは、再ページ率とは無関係に、ファイル・ページと計算ページの両方をスチールします。

maxperm
ファイル・ページによって占められる実メモリーのパーセンテージがこのレベルを上回ると、ページ置換アルゴリズムはファイル・ページのみをスチールします。

maxclient
ファイル・ページによって占められる実メモリーのパーセンテージがこのレベルを上回ると、ページ置換アルゴリズムはクライアント・ページのみをスチールします。

ファイル・ページによって占められる実メモリーのパーセンテージがパラメーター値 *minperm* と *maxperm* の間の場合、VMM は通常ファイル・ページのみをスチールしますが、ファイル・ページの再ページング率が計算ページの再ページング率より高い場合は、計算ページも同様にスチールします。

ページ置換アルゴリズムの主な目的は、計算ページが確実に公平な扱いを受けられるようにすることです。例えば、長いデータ・ファイルのメモリー内への順次読み込みによって、すぐに再使用される可能性の高いプログラム・テキストのページが失われることがあってはなりません。 ページ置換アルゴリズムがしきい値および再ページング率を使用することにより、わずかに計算ページに有利にはなりますが、両方のタイプのページが公平に扱われることが保証されます。

VMM メモリー・ロード制御機能

プロセスの実行には実メモリー・ページが必要です。 プロセスがディスク上の仮想メモリー・ページを参照する場合、それがページアウトされているか、または読み取られたことがないために、参照されるページをページインする必要があり、平均して 1 ページ以上のページがページアウトされるので、 (置換されたページが変更されている場合)、入出力トラフィックが発生し、プロセスの進行が遅れることになります。

オペレーティング・システムは、ページ置換アルゴリズムにより、近い将来参照される見込みのないページから実メモリーをスチールしようとします。 よくできたページ置換アルゴリズムは、オペレーティング・システムが、十分なプロセスをメモリー内でアクティブに保って、CPU をビジーに保つようにします。しかし、あるレベルのメモリーの競合では、近い将来すべてのページがプロセスのアクティブなプロセスによって再利用されるので、ディスクへのページアウトに適した候補となるページはありません。 この状態は以下のものによって決まります。

- システム内のメモリーの合計量
- プロセスの数
- 時間によって変化する各プロセスのメモリー所要量
- ページ置換アルゴリズム

この状態になると、ページインおよびページアウトが連続して起こります。 この状態をスラッシング といいます。 スラッシングの結果、絶え間なくページング・ディスクへの入出力が行われるため、各プロセスはディスパッチされるとすぐにページ・フォールトを検出し、どのプロセスも一向に進捗しないという結果になります。

スラッシングの最も破壊的な局面は、スラッシングがワークロードの短時間の、ランダムなピーク (例えば、システムのすべてのユーザーがたまたま同時に Enter キーを押したなど) によって引き起こされているにもかかわらず、システムは無期限に長時間スラッシングを継続する可能性があることです。

オペレーティング・システムは、メモリー・ロード制御アルゴリズムを持っており、システムがスラッシングを開始するとそれを検出して、アクティブ・プロセスを中断し、新規プロセスの始動を一定期間遅らせます。 5 つのパラメーターが、このアルゴリズムの比率およびバウンダリーを設定します。 これらのパラメーターのデフォルト値は、広範囲のワークロードにわたって「フェイルセーフ」となるように選択されています。 AIX バージョン 4 では、使用可能なメモリー・フレームの合計が 128 MB 以上となるシステムでのメモリー・ロード制御は、デフォルトで使用不可になっています。

メモリー・ロード制御アルゴリズム**:**

メモリー・ロード制御メカニズムは、1 秒当たり 1 回、アクティブなプロセスのセットに十分なメモリーが使用可能かどうかを評価します。 メモリーのオーバーコミット条件が検出されると、一部のプロセスを中断して、アクティブ・プロセスの数を減らすことによってメモリーのオーバーコミットのレベルを下げます。

プロセスが中断されると、そのスレッドはすべて、延期可能な状態に達した時点で中断されます。 中断状態のプロセスのページは直ちに失効し、ページ置換アルゴリズムによってページアウトされ、残りのアクティブ・プロセスが進行するのに十分なページ・フレームを解放します。 既存のプロセスが中断状態になっている間は、新たに作成されたプロセスも中断され、新しい作業がシステムに入るのを防ぎます。 中断状態のプロセスは、その後の間隔 (その間潜在的なスラッシング状態は存在しない) が経過するまで、再活動化はされません。 この安全な間隔が経過すると、中断状態のプロセスのスレッドは、徐々に再活動化されます。

メモリー・ロード制御 **schedo** パラメーターは、以下のものを指定します。

- システム・メモリーのオーバーコミットのしきい値 (*v_repage_hi*)
- 安全期間の作成に必要な秒数 (*v_sec_wait*)
- 個々のプロセス・メモリーのオーバーコミットしきい値。これによって、個々のプロセスが中断の候補として適格となります (*v_repage_proc*)
- プロセスを中断しようとしている時点のアクティブ・プロセスの最小数 (*v_min_process*)
- 再活動化後、あるプロセスが活動状況でいられる最小経過秒数 (*v_exempt_secs*)

これらのパラメーターの設定およびチューニングについては、 160 ページの『schedo コマンドによる VMM メモリー・ロード制御チューニング』を参照してください。

スケジューラー (プロセス 0) は 1 秒当たり 1 回、上記の測定基準すべての値を調べます。これらの値はその前の 1 秒間に収集されたもので、プロセスを中断するか活動化するかを決定します。 プロセスを中断する場合は、**-p** および **-e** パラメーター・テストにより、中断の対象となるすべてのプロセスに中断のマークが付けられます。 そのプロセスは、次にユーザー・モードで CPU を受け取るときに中断されます (中断によってアクティブ・プロセスの数が **-m** 値以下に減らない限り)。 プロセスのために重要なシステム・アクティビティーが行われている間、そのプロセスが中断の対象とならないように、ユーザー・モードの基準が適用されます。 その後の 1 秒間に、スラッシング基準がまだ満たされている場合、**-p** および **-e**

によって設定された基準を満たす追加のプロセス候補に中断のマークが付けられます。 スケジューラーがその後、安全な間隔の基準が満たされており、プロセスを再活動化すべきであると判断すると、幾つかの中断状態のプロセスが、毎秒実行キューに書き込まれ (アクティブにされ) ます。

中断状態のプロセスは、次の順に再活動化されます。

1. 優先順位
2. プロセスが中断された順序

中断状態のプロセスは、すべてが同時に再活動化されるわけではありません。 再活動化されるプロセス数の値は、その時点のアクティブ・プロセス数を認識して、その時点のアクティブ・プロセス数の 5 分の 1、または単調に増加する下限の、いずれか大きい方を再活動化するという公式によって選択されます。 この慎重なストラテジーにより、マルチプログラミングの度合いは、1 秒当たり約 20% 増加する結果となります。 このストラテジーの目的は、安全な間隔の経過後最初の 1 秒間の再活動化の比率を相対的に控えめにする一方で、その後の数秒間に再活動化の比率を着実に増やすことにあります。 プロセスの再活動化中にメモリー・オーバーコミット使用条件が再発する場合は、次の状態が起こっています。

- 再活動化が停止している。
- 再活動化すべきであるとマークされたプロセスに、再び中断状態のマークが付けられている。
- 上記の規則に従って、追加のプロセスが中断されている。

ページング・スペース・スロットの割り当てと再利用

オペレーティング・システムは、作業用ストレージについて、次のような 3 つの割り当て方法をサポートしています。

作業記憶域 (ページング・スペース・スロット とも呼ばれる) には、次のような 3 つの割り当て方法があります。

- 遅延割り当て
- 早期割り当て
- 据え置き割り当て

注**:** ページング・スペース・スロットは、プロセス (スレッドではなく) の終了または **disclaim()** システム・コールによってのみ解放されます。 このスロットは、 **free()** システム・コールによっては解放されません

遅延割り当てアルゴリズム

多くのプログラムは、最大サイズの構造体に仮想メモリーのアドレス範囲を割り当て、次に状況が必要とするだけの量の構造体のみを使用するという方法で、遅延割り当てを活用します。 アクセスされることのない仮想メモリー・アドレス範囲のページが、実メモリー・フレームまたはページング・スペース・スロットを必要とすることはありません。

この手法は、ある程度のリスクを伴います。 1 つのマシンで実行中のすべてのプログラムが、偶然最大サイズの状態に同時に遭遇すると、ページング・スペースを使い果たす可能性があります。 中には続行できず、完了に至らないプログラムもありえます。

早期割り当てアルゴリズム

この 2 番目のオペレーティング・システムのページング・スペース・スロット割り当て方式は、ご使用のシステムでこうした状態が起こりがちな場合、あるいは完了できなかったときのコストが過度に高い場合に使用するためのものです。 早期割り当てという適切な名前で呼ばれているこのアルゴリズムにより、仮想

メモリー・アドレス範囲が、例えば **malloc()** サブルーチンによって割り当てられると同時に、適切な数のページング・スペース・スロットが割り当てられます。 **malloc()** サブルーチンをサポートするだけの十分なページング・スペース・スロットがない場合は、エラー・コードが設定されます。 早期割り当てアルゴリズムは、次のように入力すると起動されます。

```
# export PSALLOC=early
```

この例では、将来すべてのプログラムが早期割り当てを使用する環境で実行されることになります。 現在実行中のシェルは影響を受けません。

早期割り当てにパフォーマンス分析者が関心を持つのは、主としてそのページング・スペース・サイズにあります。 それらのプログラムで早期割り当てをオンにすると、ページング・スペース所要量が何倍も増大することがあります。 通常推奨されるページング・スペースのサイズが、システムの実メモリー・サイズの少なくとも 2 倍であるのに対して、**PSALLOC=early** を使用するシステムの場合は、実メモリー・サイズの少なくとも 4 倍です。 実は、これは単に出発点にすぎません。 ワークロードの仮想記憶域所要量を分析し、それに合わせてページング・スペースを割り当ててください。 1 例として、AIXwindows サーバーは、早期割り当てを使用して実行する場合、250 MB のページング・スペースを必要とします。

PSALLOC=early を使用する場合、ユーザーは **sigaltstack** 機能を使ってメモリーをスタックとして事前割り当ておよび設定することにより、次の SIGSEGV シグナルのハンドラーを設定する必要があります。 **PSALLOC=early** を指定しても、ページング・スペース不足のときに、プログラムがスタックを拡張しようとすると、そのプログラムは SIGSEGV シグナルを受け取ることがあります。

据え置き割り当てアルゴリズム

3 番目のオペレーティング・システムのページング・スペース・スロット割り当て方式は、デフォルトの動作です。据え置きページ・スペース割り当て (DPSA) ポリシーは、ページング・スペースの割り当てを、そのページのページアウトが必要になるまで遅らせるので、結果としてむだなページング・スペース割り当てがなくなります。この方式は、ディスク・スペースである大容量のページング・スペースを節約します。

一部のシステムでは、アクセスされたすべてのページがタッチされていても、ページング・スペースを必要としない場合があります。 この状態が最も起こりやすいのは、非常に大容量の RAM を搭載したシステムです。 ただし、使用可能な RAM より多くの仮想メモリーがアクセスされた場合、これはページング・スペースのオーバーコミットという結果に終わることがあります。

DPSA を使用不可にし、遅延ページ・スペース割り振りポリシーを保持するには、以下のコマンドを実行します。

```
# vmo -o defps=0
```

DPSA を活動化するには、次のコマンドを実行します。

```
# vmo -o defps=1
```

一般的には、DPSA によってシステム・パフォーマンスは向上します。それは、ページ・フォールトの後、ページ・スペースを割り当てるという作業がなくなるためです。DPSA を使用すると、ページング・スペース・デバイスで必要なディスク・スペースが少なくなります。

詳細については、 168 ページの『ページ・スペース割り当て』および 107 ページの『ページング・スペースの配置およびサイズ』を参照してください。

ハード・ディスク・ストレージ管理のパフォーマンス

オペレーティング・システムはハード・ディスク・ストレージを管理するために階層構造を使用します。

物理ボリューム (PV) と呼ばれる個々のディスク・ドライブには、/dev/hdisk0 のような名前があります。物理ボリュームが使用中の場合、それはボリューム・グループ (VG) に属します。 ボリューム・グループ内の物理ボリュームはすべて、同サイズの物理パーティション (PP) に分割されます (デフォルトにより、4 GB より小さい物理ボリュームを含むボリューム・グループで 4 MB、それより大きいディスクで 8 MB 以上)。

スペース割り当ての目的で、各物理ボリュームは 5 つの領域に分割されます。 詳しくは、 217 ページの『物理ボリューム上の位置』を参照してください。 各領域内の物理パーティションの数は、ディスク・ドライブの合計容量によって変わります。

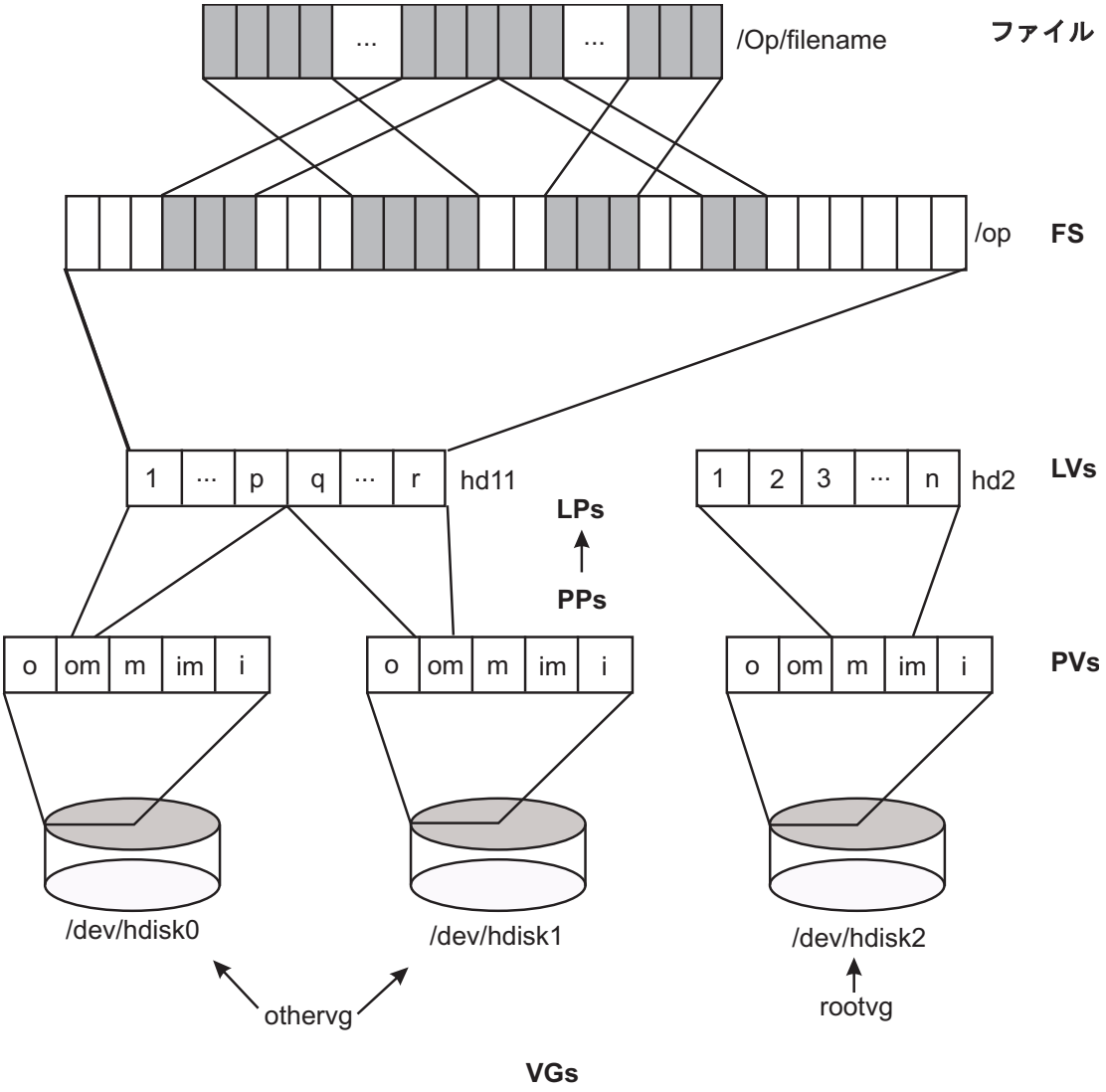


図 *10.* ハード・ディスク・データ (ミラーリングが解除された) の編成： この図は、論理ボリュームに分割された 1 つ以上の物理ボリュームの階層を示しています。 これらのパーティションまたは論理ボリュームはファイルシステムを格納し、ファイルシステムには個々のファイルを格納するディレクトリー構造があります。 ファイルは、ストレージ・メディア上のトラックに含まれるブロックに書き込まれ、通常これらのブロックは連続していません。データ・セットが消去された後、メディア上のさまざまなトラックに不規則に分散した空のブロックに新規データ・ファイルが書き込まれると、ディスクのフラグメント化が起こります。

各ボリューム・グループ内に、1 つ以上の論理ボリューム (LV) が定義されます。 各論理ボリュームは、1 つ以上のロジカル・パーティションからなっています。各ロジカル・パーティションは、少なくとも 1 つの物理パーティションに対応しています。論理ボリュームにミラーリングが指定されている場合は、各ロジカル・パーティションの追加コピーを保管するために、追加の物理パーティションが割り当てられます。 ロジカル・パーティションには連続した番号が付けられていますが、その基礎となる物理パーティションは必ずしも連続していません。

論理ボリュームは、ページングなど、システムでいろいろな目的で使用されますが、通常のシステム・データ、ユーザー・データ、またはプログラムを保持する各論理ボリュームには、単一のジャーナル・ファイルシステム (JFS または拡張 JFS) が入っています。 各 JFS は、ページ・サイズ (4096 バイト) ブロックのプールからなっています。 データがファイルに書き込まれるとき、1 つ以上の追加ブロックがそのファイルに割り当てられます。 これらのブロックは、相互に、またそのファイルに以前割り当てられたその他のブロックと、連続する場合としない場合があります。

例として、上図では、再編成せずに長期間使用中だったファイルシステムで起こりうる、悪い (ただし、考えられる最悪ではない) 状態を示しています。 `/op/filename` ファイルは、物理的に相互に離れている多数のブロックに、物理的に記録されます。 ファイルを順次に読み取ると、結果として、時間のかかる多数のシーク操作が行われます。

オペレーティング・システムのファイルは、概念的には順番に連続しているバイトであるのに対して、物理的な実体は非常に異なっている可能性があります。フラグメント化は、ファイルシステム内の割り当て/解放/再割り当てアクティビティーと同様に、論理ボリュームへの複数の拡張から生じることがあります。 使用可能なスペースが多数の小さなスペースのチャンクからなっている場合、ファイルシステムはフラグメント化されるので、新規ファイルを連続するブロックに書き込むことはできなくなります。

高度にフラグメント化されたファイルシステム内のファイルにアクセスすると、結果として多数のシークが発生し、入出力応答時間が長くなります (シーク待ち時間が入出力応答時間の大部分を占めることになります)。 例えば、ファイルが順次にアクセスされる場合、多数の、遠く離れたチャンクからなるファイルの配置は、1 つまたは少数の連続する大きいチャンクからなる配置より多くのシークを必要とします。 ファイルがランダムにアクセスされる場合、広く分散した配置には、ファイルのブロックが互いに接近している配置より長いシークが必要です。

入出力パフォーマンスにおけるファイルの配置の効果は、ファイルがメモリー内のバッファーに入ると減少します。 オペレーティング・システムでファイルを開くと、そのファイルは仮想メモリー内の永続データ・セグメントにマップされます。 このセグメントはファイルの仮想バッファーを表し、ファイルのブロックはセグメント・ページに直接マップされます。 VMM はこのセグメント・ページを管理し、ファイル・ブロックを要求に応じて (アクセスされたとき) セグメント・ページに読み込みます。 VMM がディスク上のファイルの対応するブロックに、そのページを再度書き込む原因となる状況は幾つかありますが、一般に、VMM はページが最近アクセスされた場合、そのページをメモリー内に保持します。 したがって、頻繁にアクセスされるページはメモリー内に長くとどまる傾向があり、対応するブロックにアクセスする論理ファイルは、物理ディスクにアクセスしなくても要求を満たすことができます。

ある時点で、ユーザーまたはシステム管理者は、論理ボリューム内のファイルの配置および物理ボリューム内の論理ボリュームの配置を再編成して、フラグメント化を低減し、全体の入出力ロードをより均等に分散させるという選択を行うことができます。 193 ページの『論理ボリュームおよびディスク入出力のパフォーマンス』 には、検出および修正するディスク配置およびフラグメント化の問題について、より詳しく解説されています。

固定メモリーのサポート

AIX では、常にメモリー・ページを実メモリー内に維持することができます。 このメカニズムをメモリー固定 (pinning memory) と呼びます。

メモリー領域を固定すると、固定されたメモリー領域の裏づけとなるページからのページャーによるページ・スチールが禁止されます。 システム・スペースまたはユーザー・スペースに定義されたメモリー領域を固定することが可能です。 メモリー領域が固定されると、その領域の固定がその後、解除されるまでは、その領域にアクセスしてページ・フォールトになることはありません。 カーネルの一部が固定される一方、多くの領域がページング可能で、アクセスされている間だけ固定されます。

メモリーの一部を固定した場合の利点は、固定されたページにアクセスした際に、ページ置換アルゴリズムによらずそのページを取り出すことができるという点にあります。 過剰にメモリー・ページを固定した場合は、固定していないページに対するページング・アクティビティーが増加し、パフォーマンスが低下することがあります。

vmo maxpin% チューナブルを使用すると、固定可能なメモリー量を調整することができます。 **maxpin%** チューナブルは、固定できる実メモリーの最大パーセントを指定します。

注**:** カーネルはカーネル・データの一部の量を固定できなければならないため、**maxpin%** チューナブルを減らすと機能上の問題が生じる可能性があるため、お勧めできません。

ユーザー・アプリケーションは幾つかの異なるメカニズムを通じてメモリーを固定する可能性があります。アプリケーションはアプリケーション・メモリーを固定するために、**plock()**、**mlock()**、および **mlockall()** サブルーチンを使用することができます。

アプリケーションは、**SHM_LOCK** オプションを **shmctl()** サブルーチンに指定することにより、共有メモリー領域を明示的に固定することができます。 またアプリケーションは、**shmget()** に **SHM_PIN** フラグを指定しても、共有メモリー領域を固定することができます。

マルチプロセッシング

どの時点においても、シングル・プロセッサー・チップが作動可能な速度には技術的な制限があります。システムのワークロードをシングル・プロセッサーでは十分に処理できない場合、1 つの答えは、その問題に複数のプロセッサーを適用することです。

この答えが成功するかどうかは、システムの設計者のスキルだけでなく、ワークロードがマルチプロセッシングに適しているかどうかにも左右されます。人が行う作業に置き換えてみると、人数を追加することは、その作業がフリーダイヤル番号の呼び出しに応答することであれば、よいアイデアといえるかもしれませんが、車を運転するといった作業の場合は、効果は疑わしいものです。

ユニプロセッサーからマルチプロセッサー・システムへの移行計画の目標がパフォーマンスの向上にある場合は、以下の条件が当てはまらなければなりません。

- ワークロードが使用するプロセッサーが限定され、そのユニプロセッサー・システムが飽和状態にある。
- ワークロードが、トランザクションや複雑な計算など、プロセッサーを集中的に使用する複数のエレメントを含み、それらが同時に、または独立して実行できる。
- 既存のユニプロセッサーをアップグレードすることも、適切な能力のある別のユニプロセッサーに取り替えることもできない。

未変更の単一スレッド・アプリケーションは、通常マルチプロセッサー環境でも正しく機能しますが、そのパフォーマンスは予期しない変化を示すことがよくあります。 マルチプロセッサーへの移行により、システムのスループットが改善され、複雑なマルチスレッド・アプリケーションの実行時間も改善されますが、個々の単一スレッド・コマンドの応答時間が改善されることはまずありません。

マルチプロセッサー・システムから可能な最高のパフォーマンスを得るには、マルチプロセッサー環境に固有のオペレーティング・システムとハードウェア実行のダイナミックスについて理解しておく必要があります。

対称型マルチプロセッサーの概念およびアーキテクチャー

システムの複雑さを増す何らかの変更を行うと共に、複数のプロセッサーを使用すると、満足のいく操作およびパフォーマンスのために取り組まなければならない設計上の考慮事項が生じます。

複雑さが増したことによって、ハードウェア/ソフトウェアのトレードオフに関する範囲が広がり、ユニプロセッサー・システムの場合より、ハードウェア/ソフトウェアの設計の調整を精密に行う必要があります。 設計上の問題の答えとトレードオフとのさまざまな組み合わせにより、多種多様なマルチプロセッサー・システム体系が生まれます。

マルチプロセッシングのタイプ

マルチプロセッシング (MP) システムには幾つかのカテゴリーが存在します。

非共用 **MP:**

プロセッサー間では何も共用していませんが (それぞれが独自のメモリー、キャッシュ、およびディスクを備えている)、プロセッサーは相互接続されています。 このタイプのマルチプロセッシングは「純正クラスター」とも呼ばれています。

各プロセッサーは完全なスタンドアロン・マシンで、オペレーティング・システムのコピーを実行します。 LAN 接続の場合、プロセッサーは疎結合です。 スイッチで接続されている場合は、プロセッサーは密結合です。 プロセッサー間の通信は、メッセージ・パッシングを通じて行われます。

このようなシステムの利点は、スケーラビリティーが非常に優れていることと、高可用性です。こうしたシステムの欠点は、プログラミング・モデル (メッセージ・パッシング) がよく知られていないことです。

共用ディスク **MP:**

共用ディスクの利点は、よく知られているプログラミング・モデルの一部がそのまま残されている (ディスク・データはアドレス可能で整合性があり、メモリーはそうではない) ことと、高可用性が共有メモリー・システムに比べてはるかに容易に得られることです。 欠点は、共用データへの物理アクセスと論理アクセスにおけるボトルネックのために、スケーラビリティーが限定されることです。

プロセッサーは独自のメモリーおよびキャッシュを備えています。 プロセッサーは並列で実行され、ディスクを共用します。 各プロセッサーはオペレーティング・システムのコピーを実行し、プロセッサー間は疎結合 (LAN を通じて接続) です。 プロセッサー間の通信は、メッセージ・パッシングを通じて行われます。

共有メモリー・クラスター**:**

共有メモリー・クラスター内のプロセッサーはすべて、独自のリソース (メイン・メモリー、ディスク、入出力) を持ち、各プロセッサーがオペレーティング・システムのコピーを実行します。

プロセッサーは密結合 (スイッチを通じて接続) です。プロセッサー間の通信は、共有メモリーを通じて行われます。

共有メモリー **MP:**

すべてのプロセッサーは、同じ筐体内で、高速バスまたはスイッチにより密結合されています。 プロセッサーは、同じグローバル・メモリー、ディスク、および入出力装置を共用します。 オペレーティング・システムの 1 つのコピーだけが、すべてのプロセッサーにわたって実行されるので、オペレーティング・システムはこのアーキテクチャーを活用するように設計されていなければなりません (マルチスレッド化オペレーティング・システム)。

SMP には、次のような幾つかの利点があります。

- SMP はスループットを増やすための費用対効果の高い方法である。
- 単一システム・イメージで実行され、オペレーティング・システムがすべてのプロセッサー間で共用される (管理が容易)。
- 複数プロセッサーを 1 つの問題に適用する (並列プログラミング)。
- ロード・バランシングはオペレーティング・システムが行う。
- ユニプロセッサー (UP) プログラミング・モデルを SMP で使用することができる。
- 共用データについてスケーラブルである。
- すべてのデータがすべてのプロセッサーによってアドレス可能であり、ハードウェア・スヌープ・ロジックによって整合性が保たれる。
- 通信がグローバル共有メモリーを通じて行われるので、プロセッサー間の通信にメッセージ・パッシング・ライブラリーを使用する必要がない。
- システムにプロセッサーを追加することによって、処理能力の問題を解決することができる。 ただし、SMP システムにさらにプロセッサーを追加する場合のパフォーマンスの向上については、現実的な予想を設定する必要があります。
- 現在、ますます多くのアプリケーションおよびツールが使用可能になっている。 ほとんどの UP アプリケーションは SMP アーキテクチャーで実行できるか、または SMP アーキテクチャーに移植することができる。

SMP システムには、次のような制限があります。

- キャッシュの整合性、ロック機構、共用オブジェクトその他のために、スケーラビリティーに制限がある。
- マルチプロセッサーを活用するには、スレッドのプログラミングやデバイス・ドライバーのプログラミングなどの新しいスキルが必要になる。

アプリケーションの並列化

アプリケーションは SMP 上で、2 つの方法のうちのいずれかを使用して並列化することができます。

- 従来の方法は、アプリケーションを複数のプロセスに分割するというものでした。これらのプロセスは、パイプ、セマフォー、または共有メモリーなどのプロセス間通信 (IPC) を使用して通信します。各プロセスは、他のプロセスからのメッセージなどのイベント待ちをブロックできなければなりませんが、同時に、ロックなどを使用して共用オブジェクトへのアクセスを調整する必要があります。
- もう 1 つの方法は、UNIX (POSIX) スレッド用の移植可能なオペレーティング・システム・インターフェースを使用する方法です。 スレッドにはプロセスと同様の調整上の問題があり、その問題に対処す

るための類似のメカニズムを持っています。 したがって、単一のプロセスは、任意の数のスレッドを、別々のプロセッサーで同時に実行することができます。 スレッドを調整し、共用データへのアクセスをシリアライズするのは、開発者の責任です。

アプリケーションを並列化する際にどちらの方法を使用するか決定する場合は、スレッドとプロセスの両方の利点を考慮してください。 スレッドの方がプロセスより高速であり、しかも簡単にメモリーを共用できます。 これに対して、プロセスをインプリメントすると、複数のマシンまたはクラスターに簡単に分散できるようになります。 アプリケーションが新規インスタンスを作成または削除する必要がある場合は、スレッドの方が高速です (fork プロセスでのオーバーヘッドが多い)。 その他の機能については、スレッドのオーバーヘッドはプロセスのオーバーヘッドとほぼ同じです。

データ・シリアライゼーション

複数のスレッドによる読み取りまたは書き込みが可能な記憶素子はどれも、プログラムの実行中に変更されることがあります。

これは一般に、マルチプロセッシング環境と同様マルチプログラミング環境でもいえることですが、マルチプロセッサーの出現により、この考慮事項の有効範囲および重要性が、次の 2 つの点で増大しています。

- マルチプロセッサーおよびスレッド・サポートにより、スレッド間でデータを共有するアプリケーションの作成が魅力あるものになり、また容易になった。
- カーネルは、もう単に割り込みを使用不可にすることによって、シリアライゼーションの問題を解決することはできない。

注: 重大な問題を回避するために、データを共用するプログラムは、そのデータを並列にではなく、シリアルにアクセスするよう配置する必要があります。 プログラムでは、共用データ項目を更新する前に、他のプログラム (別のスレッドで実行中の、そのプログラム自体の別のコピーも含む) がその項目を変更しないようにする必要があります。 読み取りは通常並列に行うことができます。

プログラムが相互に干渉し合わないようにするために使用される基本のメカニズムがロック です。 ロックとは、1 つ以上のデータ項目へのアクセスを許可することを表す抽象概念です。 ロックおよびアンロック要求はアトミックです。つまり、これらの要求は、割り込みもマルチプロセッサー・アクセスもその結果に影響を及ぼさないような方法でインプリメントされます。 共用データ項目にアクセスするすべてのプログラムは、そのデータ項目を操作する前に、そのデータ項目に対応するロックを取得する必要があります。そのロックが既に別のプログラムによって保持されている場合 (または別のスレッドが同じプログラムで実行中の場合)、要求側のプログラムは、ロックが使用可能になるまでそのアクセスを遅らせる必要があります。

ロックを待つために時間を費やす上に、シリアライゼーションによって、スレッドがディスパッチ不可となる回数が増加します。 スレッドがディスパッチ不可となっている間に、他のスレッドは多くの場合、ディスパッチ不可のスレッドのキャッシュ線が置き換えられる原因となり、その結果、スレッドが最終的にロックを入手してディスパッチされるときの、メモリー待ち時間のコストは増大します。

オペレーティング・システムのカーネルには多数の共用データ項目が含まれるので、内部的にシリアライゼーションを行う必要があります。 したがって、他のプログラムとデータを共用していない 1 つのアプリケーション・プログラム内でも、シリアライゼーション遅延が生じることがあります。これは、このプログラムによって使用されるカーネル・サービスが、共用カーネル・データをシリアライズしなければならないからです。

ロック

オペレーティング・システムの内部メモリーを割り当てたりフリーにしたりする場合にロックを使用します。

詳しくは、『ロックの理解 (Understanding Locking)』を参照してください。

ロックのタイプ:

オープン・ソフトウェア・ファウンデーション/1 (OSF/1) 1.1 ロック方法論が、 AIX マルチプロセッサー・ロック機能用モデルとして使用されました。

しかし、システムはプリエンプタブルかつページング可能であるため、OSF/1 1.1 ロック・モデルに幾つかの特性が追加されました。 単純ロックおよび複合ロックはプリエンプタブルです。 また、ビジーな単純ロックを獲得しようとしたときに、ロックのオーナーが現在実行中でなければ、スレッドがスリープ状態に入る可能性もあります。 さらに、プロセッサーが一定の時間 (この時間値はシステム全体での変数)、単純ロックでスピンしている場合、単純ロックはスリープ・ロックに変わります。

単純ロック:

AIX オペレーティング・システム バージョン 4 の単純ロックは、特定の条件下でスレッドが無期限にスピンしないようにスリープする、スピン・ロックです。

単純ロックはプリエンプタブルですが、これは、カーネル・スレッドを、別のより優先順位の高いカーネル・スレッドが、そのスレッドが単純ロックを保持している間は優先使用できることを意味します。 マルチプロセッサー・システムでは、スレッド - 割り込みクリティカル・セクションを保護する単純ロックは、実行中のプロセッサー内および異なるプロセッサー間の両方で実行をシリアライズするために、割り込み制御とともに使用する必要があります。

ユニプロセッサー・システムでは、割り込み制御で十分であり、ロックを使用する必要はありません。 単純ロックは、スレッド - スレッドおよびスレッド - 割り込みクリティカル・セクションを保護するためのものです。 単純ロックは、割り込みハンドラー内にある場合、ロックが使用可能になるまでスピンします。 単純ロックには、ロック状態とアンロック状態の 2 つの状態があります。

複合ロック:

AIX の複合ロックは、スレッド - スレッドのクリティカル・セクションを保護する読み取り/書き込みロックです。 これらのロックはプリエンプタブルです。

複合ロックは、特定の条件下でスリープするスピン・ロックです。デフォルトでは、複合ロックは再帰的ではありませんが、**lock_set_recursive()** カーネル・サービスを通じて再帰的となることができます。 複合ロックには、排他書き込み、共用読み取り、またはアンロック状態の 3 つの状態があります。

ロックの細分性:

マルチプロセッサー環境で作業するプログラマーは、共用データのために別々のロックをいくつ作成する必要があるか決定しなければなりません。 一組の共用データ項目全体をシリアライズする単一ロックが存在する場合は、ロック・コンテンションは比較的起こりやすくなります。 広く使用されるロックの存在により、システムのスループットに上限が設けられます。

別個の各データ項目が独自のロックを持っていれば、2 つのスレッドがそのロックを求めて競合する可能性は比較的低くなります。 ただし、追加の各ロック・コールおよびアンロック・コールにはプロセッサー時間がかかり、複数ロックの存在により、デッドロックを起こす可能性もあります。 最も単純な例で、デッ

ドロックとは以下の図に示すような状態です。ここで、スレッド 1 はロック A を所有し、ロック B を待っています。それに対し、スレッド 2 はロック B を所有し、ロック A を待っています。どちらのプログラムも、デッドロックを中断する **unlock()** コールに達することはありません。 デッドロックの通常の予防法は、ロックの所定のセットを使用するすべてのプログラムが、常にまったく同じ順序でそれらのロックを獲得しなければならないようなプロトコルを確立することです。

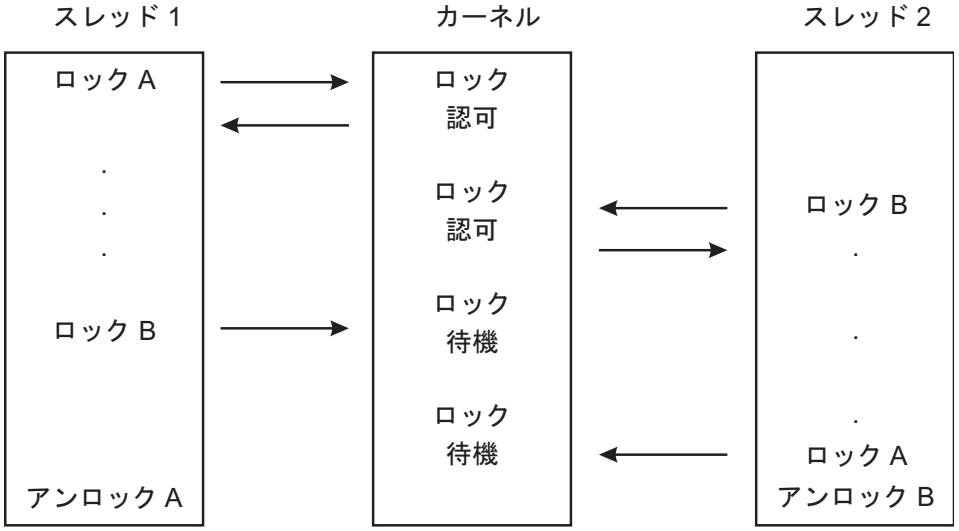


図 *11.* デッドロック： この図に示されているデッドロックでは、スレッド 1 の列はロック A を所有し、ロック B を待っています。それに対し、スレッド 2 の列はロック B を所有し、ロック A を待っています。どちらのプログラム・スレッドも、デッドロックを中断する unlock コールに達することはありません。

キューイング理論に従って、リソースのアイドル時間が少ないほど、そのリソースを取得するための平均待ち時間が長くなります。 両者の関係は正比例するわけではなく、ロックが倍増すると、そのロックの平均待ち時間は 2 倍より長くなります。

ロックの待ち時間を低減する最も有効な方法は、そのロックが保護している内容のサイズを縮小することです。 以下にいくつかのガイドラインを示します。

- ロックが要求される頻度を減らす。
- コンポーネント内のすべてのコードではなく、共用データにアクセスするコードだけをロックする (これにより、ロックを保持する時間が減少します)。
- ルーチン全体ではなく、特定のデータ項目や構造体だけをロックする。
- ロックは、ルーチンにではなく、常に特定のデータ項目や構造体に関連付ける。
- 大型のデータ構造体の場合、構造体全体に対して 1 つのロックを選択するのではなく、構造体のエレメントごとに 1 つずつロックを選択する。
- ロックを保持している間は、同期入出力その他のブロッキング・アクティビティーを決して行わない。
- 所定のコンポーネント内の同じデータに複数回アクセスする場合は、ロック・アンロックの 1 回のアクションで扱えるように、複数のアクセスをまとめて移動するようにする。
- ウェイクアップを 2 回行わない。 ロック中のデータの一部を変更し、そのことを誰かに通知する必要がある場合は、そのウェイクアップを通知する前に、ロックを解放してください。
- 同時に 2 つのロックを保持しなければならない場合は、最もビジーなロックを最後に要求する。

他方では、細分性の度合いが細かすぎるために、ロック要求およびロック解放の頻度が増加し、さらに命令が追加されることになります。 そこで、細かすぎる細分性と粗すぎる細分性の間の平衡点を見つける必要があります。 最適の細分性を見つけるには試行錯誤が必要で、これは MP システムにおける大きな課題の 1 つです。 以下のグラフは、スループットとロックの細分性の関係を示しています。

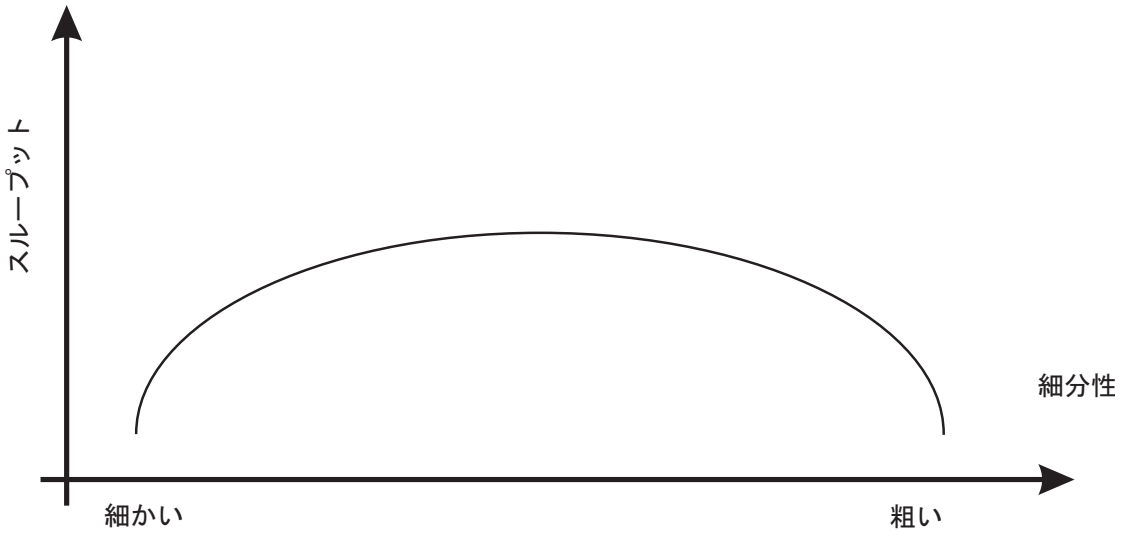


図 *12.* スループットと細分性の関係： この図は単純な 2 軸のグラフです。 縦軸 (y 軸) はスループットを表します。 横軸 (x 軸) は細分性の度合いを表し、スケールの外側に進むにつれて細かい細分度から粗くなっていきます。引き延ばされたベル状の曲線は、スループットに対する細分性の度合いの関係を示しています。細かい細分度から粗い細分度に変化するにつれて、スループットは徐々に最大レベルまで増大し、その後ゆっくりと低下し始めます。 このことは、最大のスループットを得るためには細分度を中程度にする必要があることを示しています。

ロックのオーバーヘッド**:**

ロック要求、ロック待機、およびロック解除は、処理オーバーヘッドを増やします。

- マルチプロセッシングをサポートするプログラムは、ユニプロセッサーで実行していようと、あるいはマルチプロセッサー・システムにおける問題のロックの唯一のユーザーであろうと、常に同じロック処理およびアンロック処理を行う。
- あるスレッドが、別のスレッドが保持するロックを要求する場合、要求側のスレッドはしばらくスピンするかまたはスリープ状態になり、可能なら、別のスレッドをディスパッチする。 これはプロセッサー時間を消費します。
- 広く使用されるロックの存在により、システムのスループットに上限が設けられる。 例えば、所定のプログラムが相互排他ロックの保持にその実行時間の 20% を費やす場合、システム内のプロセッサー数にかかわらず、そのプログラムの最大 5 つのインスタンスを同時に実行することができます。実際は 5 つのインスタンスでさえ、多くの場合、相互に待ちを回避できるほど正確に同期することはありません (71 ページの『マルチプロセッサーのスループットのスケーラビリティー』を参照してください)。

ロック待機**:**

スレッドが既に別のスレッドに所有されているロックを必要とする場合、そのスレッドはブロックされ、ロックが解除されるまで待つ必要があります。

待機には次の 2 つの方法があります。

- きわめて短時間しか保持されないロックの場合は、スピン・ロックが適している。スピン・ロックにより、待ち状態のスレッドはそのプロセッサーを保持し、ロックが使用可能になるまで緊密なループ (スピン) 内でロック・ビットを繰り返し調べることができます。 スピンの結果、CPU 時間が増加します (カーネルまたはカーネル・エクステンション・ロックのシステム時間)。
- 長時間保持されるロックの場合は、スリープ・ロックが適している。スレッドはロックが解除されるまでスリープ状態に入り、ロックが使用可能になると実行キューに戻されます。 スリープの結果、アイドル時間が長くなります。

待機が発生すると、常にシステム・パフォーマンスが低下します。 スピン・ロックを使用すると、プロセッサーはビジーになりますが、有効な処理を実行しません (スループットには関係ありません)。スリープ・ロックを使用すると、コンテキストの切り替えとディスパッチによるオーバーヘッドが発生し、その結果、キャッシュのミスが増加します。

オペレーティング・システム開発者は、次の 2 つのロックのいずれかを選択することができます。「相互に排他的な単純ロック」の場合は、ロックが使用可能になるのを待つ間、プロセスはスピンおよびスリープします。「複雑な読み取り/書き込みロック」の場合は、ロックが使用可能になるのを待つ間、プロセスはスピンおよびブロックします。

ロックの使用に関する規則は厳密に定義されています。 ハードウェアにもソフトウェアにも、強制メカニズムまたは検査メカニズムはありません。 ロックを使うことで、AIX バージョン 4 は「MP Safe」になりましたが、その反面、開発者は独自のグローバル・データを保護するために、適切なロック・ストラテジーを定義したりインプリメントしたりする責任があります。

キャッシュの整合性

マルチプロセッサーの設計において、技術者は、キャッシュの整合性を確保することに、相当な注意を払います。 成功しても、キャッシュの整合性にはパフォーマンス・コストがかかります。

取り組むべき問題を理解する必要があります。

各プロセッサーがメモリーのさまざまな部分の状態を反映するキャッシュを 1 つずつ備えている場合、複数のキャッシュが同じ線のコピーを持っている可能性があります。 また、所定の線に複数のロック可能データ項目が含まれている可能性もあります。 2 つのスレッドがそれらのデータ項目に、適切にシリアライズされた変更を加えた場合、結果として、両方のキャッシュに別々の、誤ったバージョンのメモリーの線が含まれることになる場合があります。 言い換えれば、メモリーの特定領域の内容と思われているものについて、システムに 2 つの異なるバージョンが含まれているため、システムの状態にもう整合性はありません。

キャッシュの整合性の問題は、通常、行が変更された場合、重複する行の 1 つを除くすべてを無効にすることによって解決できます。 ハードウェアは、線を無効にする際、ソフトウェアの介入なしにスヌープ・ロジックを使用しますが、キャッシュ線を無効にされたどのプロセッサーも、次にその線がアドレスされると、キャッシュ・ミスとそれに伴う遅延を生じます。

スヌープ とは、キャッシュの整合性の問題を解決するために使用されるロジックのことです。プロセッサーのスヌープ・ロジックは、そのキャッシュ内のワードが変更されるたびに、バスを通じてメッセージをブロードキャストします。 また、スヌープ・ロジックは、他のプロセッサーからのこの種のメッセージを探して、バス上をスヌープします。

プロセッサーが、その専用キャッシュに存在するアドレスの値を別のプロセッサーが変更したことを検出すると、スヌープ・ロジックは、そのキャッシュ内のエントリーを無効にします。 これは相互無効化 *(cross invalidate)* と呼ばれます。 相互無効化によって、プロセッサーは、キャッシュ内の値が無効であること

と、どこか別の場所 (メモリーまたはその他のキャッシュ) で正しい値を探す必要があることを認識します。 相互無効化によってキャッシュ・ミスが増え、さらにスヌープ・プロトコルによるバス・トラフィックが加わるため、キャッシュの整合性の問題を解決すると、すべての SMP のパフォーマンスおよびスケーラビリティーは低下します。

プロセッサーとの親和性およびバインディング

プロセッサーとの親和性 *(processor affinity)* とは、以前スレッドを実行していた同じプロセッサーに、そのスレッドをディスパッチできる可能性のことです。 プロセッサーとの親和性における強調の度合いは、スレッドのキャッシュ作業セットのサイズに正比例し、スレッドが最後にディスパッチされた後の時間の長さに反比例して変動します。 AIX バージョン 4 のディスパッチャーはプロセッサーとの親和性を強化されており、親和性はオペレーティング・システムによって暗黙的に行われます。

スレッドに割り込みが行われ、後で同じプロセッサーにディスパッチし直された場合、プロセッサーのキャッシュにはまだ、そのスレッドに属する線が含まれている可能性があります。スレッドは、異なるプロセッサーにディスパッチされると、多くの場合、そのスレッドのキャッシュ作業セットが RAM または他のプロセッサーのキャッシュから検索されるまで、一連のキャッシュのミスを経験することになります。 他方、ディスパッチ可能スレッドが、以前実行されていたプロセッサーが使用可能になるまで待機しなければならない場合、そのスレッドにはさらに長い遅延が生じることになります。

最も強力な形のプロセッサー親和性は、スレッドを特定のプロセッサーにバインドすることです。 バインディングとは、他のプロセッサーが使用可能かどうかにかかわらず、スレッドを、そのバインドしたプロセッサーのみにディスパッチすることです。 **bindprocessor** コマンドおよび **bindprocessor()** サブルーチンは、指定されたプロセスのスレッド (複数の場合もある) を特定のプロセッサーにバインドします (84 ページの『bindprocessor コマンド』を参照してください)。 明示的バインディングは、**fork()** および **exec()** システム・コールによって継承されます。

バインディングは、ほとんど割り込みが起こらない CPU 集中プログラムに役立つ場合もあります。 通常のプログラムの場合、バインディングは時には逆効果になります。これは、入出力後、スレッドがバインドされるプロセッサーが使用可能になるまで、スレッドの再ディスパッチを遅らせる可能性があるからです。スレッドが入出力操作の間中ブロックされている場合、その処理コンテキストの多くが、スレッドがバインドされているプロセッサーのキャッシュに残っているとは考えられません。 多くの場合、スレッドにとっては次に使用可能なプロセッサーにディスパッチされた方がよいサービスを受けられます。

メモリーおよびバスのコンテンション

ユニプロセッサーでは、メモリーおよび入出力のバンクやメモリー・バスなどの一部の内部リソースに対するコンテンションは、通常、時間を費やすマイナー・コンポーネントです。 マルチプロセッサーでは、特にキャッシュ整合性アルゴリズムにより RAM へのアクセス数が増えた場合、これらの影響はさらに重大になります。

SMP のパフォーマンス上の問題

SMP を効果的使用するために注意することが幾つかあります。

ワークロードの並行性

SMP システムに固有の、主なパフォーマンス上の問題は、ワークロードの並行性です。これは、次のように表現することができます。「*n* 台のプロセッサーがある以上、これをすべて有効に使用したいが、どうすればよいだろうか。」

4-way マルチプロセッサー・システムでも、常時、実質的な作業を行うプロセッサーが 1 台だけだとすれば、ユニプロセッサーと同じことです。さらに、プロセッサー間で干渉が起こらないように追加のコードが必要なため、ユニプロセッサーより悪い場合もあります。

ワークロード並行性は、シリアライゼーションを補完するものです。 システム・ソフトウェアまたはアプリケーション・ワークロード (またはその 2 つの相互作用) がシリアライゼーションを必要とする限り、ワークロード並行性も低下します。

また、ワークロード並行性はプロセッサーとの親和性が増すことによっても、より望ましい形で低下します。 プロセッサーとの親和性によって得られるキャッシュ効率の向上の結果、プログラムはより高速に完了します。 ワークロード並行性は低下しますが (ディスパッチ可能スレッドがさらに使用可能でない限り)、応答時間は向上します。

ワークロード並行性のコンポーネントであるプロセス並行性 とは、マルチスレッド・プロセスがいつでも複数のディスパッチ可能スレッドを持っている度合いです。

スループット

SMP システムのスループットは、大抵の場合、さまざまな要因に依存しています。

- 一貫して高水準のワークロード並行性。 ある時間にディスパッチ可能なスレッドの数がプロセッサーの数より多かったとしても、それらのスレッドを別の時間にアイドルのプロセッサーで処理することはできません。
- ロック・コンテンションの量。
- プロセッサー親和性の程度。

応答時間

SMP システム内の特定プログラムの応答時間は、幾つかの要因に依存しています。

- プログラムのプロセス並行性レベル。 プログラムが一貫して複数のディスパッチ可能スレッドを持っている場合、その応答時間は多くの場合、SMP 環境では向上します。 プログラムが単一スレッドからなっている場合、その応答時間は、最もよくて同じ速度のユニプロセッサーの応答時間と同等程度です。
- 当該プログラムのその他のインスタンスのロック・コンテンション、または同じロックを使用する他のプログラムとのロック・コンテンションの量。
- プログラムのプロセッサーとの親和性の程度。 プログラムの各ディスパッチが、そのプログラムのキャッシュ線をまったく持たない別のプロセッサーに対して行われる場合、プログラムの実行速度は、同程度のユニプロセッサーで実行するより遅くなることがあります。

SMP ワークロード

プロセッサーの追加がパフォーマンスに及ぼす影響より、処理されている特定ワークロードのある種の特性による影響のほうが大きくなります。 このセクションでは、それらの重要な特性と、その影響について説明します。

以下の用語は、SMP 環境で操作するために、既存のプログラムが変更されている範囲、または新規プログラムが設計されている範囲を説明する際に使用されます。

SMP safe (SMP セーフ)
: プログラムの中で、共用データへのシリアライズされていないアクセスなど、SMP 環境において機能上の問題の原因となるようなアクションをすべて回避すること。 この用語を単独で使用した場合は、通常、SMP 環境で正しく機能するために必要最小限の変更だけが加えられたプログラムを指します。

SMP efficient (SMP 効率)
プログラムの中で、SMP 環境において機能上またはパフォーマンス上の問題の原因となるようなアクションをすべて回避すること。 SMP-efficient と記述されているプログラムは SMP-safe でもあります。 SMP-efficient プログラムには通常、初期のボトルネックを最小化するために追加の変更が加えられています。

SMP exploiting (SMP 活用)
特に SMP 環境を有効に使用することを意図して、マルチスレッド化などの機能をプログラムに追加すること。 SMP-exploiting と記述されているプログラムは、一般に SMP-safe かつ SMP-efficient でもあると見なされます。

ワークロードのマルチプロセッシング

高速のコンピューターで多量のワークロードを実行するマルチプログラミング・オペレーティング・システムは、人間の感覚には、幾つかのことが同時に起こっているという印象を与えます。

実際は、多くの要求側ワークロードは、どの瞬間においても多数のディスパッチ可能スレッドを持っているわけではありません。これは、シリアライゼーションがそれほど問題にならないシングル・プロセッサー・システムで実行する場合も同じです。 常に少なくともプロセッサーと同じ数のディスパッチ可能スレッドが存在するのでなければ、1 つ以上のプロセッサーがアイドルになる時間があります。

ディスパッチ可能スレッドの数は、システム内のスレッドの合計数から以下を引いたものです。

- 入出力待ちのスレッドの数
- 共用リソース待ちのスレッドの数
- 別のスレッドの結果待ちのスレッドの数
- スレッド自体の要求でスリープ中のスレッドの数

ワークロードは、システム内にいつでもプロセッサーと同じ数のディスパッチ可能スレッドがある限り、マルチプロセッシング可能ということができます。これは単にディスパッチ可能スレッドの平均数がプロセッサー数と等しいことを意味するのではないことに注意してください。 その時間の半分はディスパッチ可能スレッドの数がゼロで、残りの時間はプロセッサー数の 2 倍の数だとすると、ディスパッチ可能スレッドの平均数はプロセッサー数と等しくなりますが、システム内のある特定のプロセッサーはどれも、その時間の半分しか作動しません。

ワークロードのマルチプロセッシングの可能性を高めると、必然的に以下のいずれか、または両方が関係することになります。

- スレッドの待機の原因となるボトルネックの識別および解決
- システム内のスレッドの合計数の増加

これらの解決策は独立していません。 単一の主要なシステム・ボトルネックがある場合、そのボトルネックを通過する既存のワークロードのスレッド数を増やすと、単に待機中のスレッドの比率が増加します。現在ボトルネックが存在しない場合でも、スレッドの数を増やすと、ボトルネックが作成されます。

マルチプロセッサーのスループットのスケーラビリティー

実際のワークロードは、SMP システムでは完全にはスケーリングしません。

完全なスケーリングを阻む要因には、次のものがあります。

- プロセッサー数が増加すると同時に、バス/スイッチ・コンテンションが増加する。
- メモリー・コンテンションが増加する (すべてのメモリーがすべてのプロセッサーによって共用される)。

- メモリーがさらに少なくなるにつれて、キャッシュ・ミスのコストが増加する。
- キャッシュの整合性を維持するための、キャッシュの相互無効化および別のキャッシュからの読み取り。
- ディスパッチング率の上昇によるキャッシュ・ミスの増加 (システムでさらに多くのプロセス/スレッドをディスパッチする必要があるため)。
- 同期命令のコストの増加。
- オペレーティング・システムおよびアプリケーション・データ構造体の大型化によるキャッシュ・ミスの増加。
- ロック・アンロックのためのオペレーティング・システムおよびアプリケーションのパス長さの増加。
- ロック待ちのためのオペレーティング・システムおよびアプリケーションのパス長さの増加。

上記の要因はすべて、ワークロードのスケーラビリティー と呼ばれるものに寄与します。 スケーラビリティーとは、プロセッサーの追加によってワークロードのスループットが改善される度合いです。 これは通常、マルチプロセッサー上のワークロードのスループットを、同等のユニプロセッサー上のスループットで割った商で表されます。 例えば、ユニプロセッサーが所定のワークロードで 1 秒当たり 20 の要求を達成し、 4 台のプロセッサーからなるシステムが 1 秒当たり 58 の要求を達成した場合、スケール・ファクターは 2.9 になります。 このワークロードは非常にスケーラブルです。 ワークロードの中でも、長時間時実行され、入出力またはその他のカーネル・アクティビティーを無視してよく、共用データのない、計算主体のプログラムのみからなるワークロードは、4 台のプロセッサーを使用するシステムでのスケール・ファクターはほぼ 3.2 から 3.9 になると考えられます。 しかし、現実にはワークロードの大部分はこの値に達しません。 スケーラビリティーの見積もりは非常に難しいので、スケーラビリティーを仮定する場合は、本当のワークロードの測定を基にして行うべきです。

次の図は、スケーリングの問題を示しています。 ワークロードは、仮定的な一連のコマンドからなっています。 各コマンドの 3 分の 1 は通常処理、3 分の 1 は入出力待ち、3 分の 1 はロック保留の処理です。 ユニプロセッサーでは、ロックが保留されているかどうかにかかわらず、実際に処理を行えるのは一度に 1 つのコマンドだけです。 示されている時間間隔 (コマンドの単体実行時の 5 倍) に、ユニプロセッサーは 7.67 のコマンドを処理します。

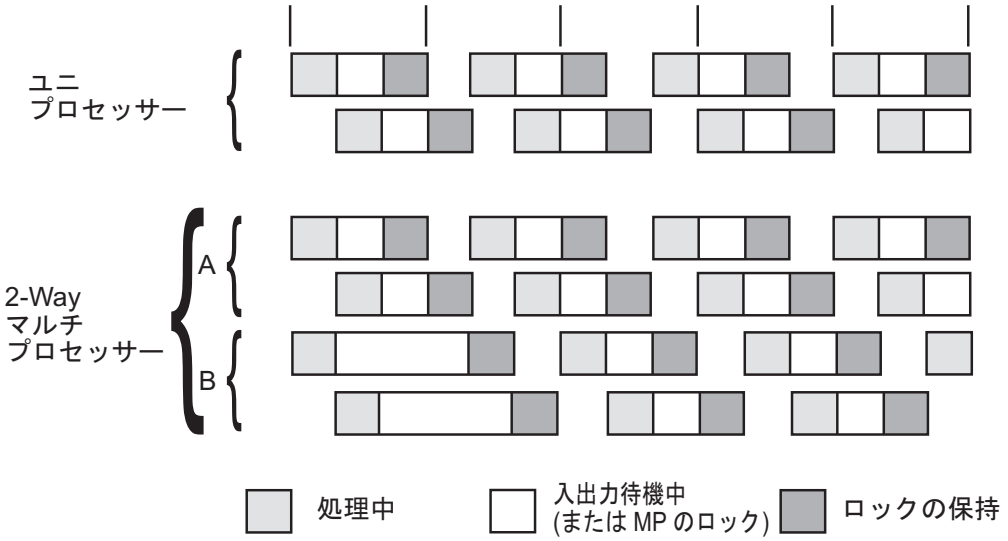


図 *13.* マルチプロセッサーのスケーリング： この図は、スケーリングの問題を示しています。 ワークロードは、仮定的な一連のコマンドからなっています。 各コマンドの 3 分の 1 は通常処理、3 分の 1 は入出力待ち、3 分の 1 はロック保留の処理です。 ユニプロセッサーでは、ロックが保留されているかどうかにかかわらず、実際に処理を行えるのは一度に 1 つのコマンドだけです。 同じ時間間隔に、ユニプロセッサーは 7.67 個のコマンドを処理しますが、マルチプロセッサーは 14 のコマンドを処理するので、スケール・ファクターは 1.83 です。

マルチプロセッサーでは、2 台のプロセッサーがプログラム実行を処理しますが、ロックは 1 つしか存在しません。 簡単にするために、プロセッサー B に影響を及ぼすロック・コンテンションはすべて示されています。示されている期間内に、マルチプロセッサーは 14 のコマンドを処理します。 したがって、スケール・ファクターは 1.83 です。それ以上の台数の例を示しても状態は変わらないので、ここでは 2 台のプロセッサーの例にとどめました。ここで、ロックはその時間の 100% 使用中になっています。 4 台のプロセッサーを使用するマルチプロセッサーでは、スケール・ファクターは 1.83 以下です。

実際のプログラムが上の図のコマンドほど対称になることは、まずありません。 さらに、ここではコンテンションの 1 つの次元、つまり、ロックしか考慮に入れていません。 キャッシュ整合性およびプロセッサーとの親和性の影響を含めていれば、スケール・ファクターはほぼ確実にこれより低い値になるでしょう。

この例では、単にプロセッサーを追加しただけで、高速に実行できなくなることがよくあるワークロードを図示しています。 さらに、スレッド間のコンテンションのソースを識別し、最小化することも必要です。

スケーリングはワークロードに依存します。 一部の公表されたベンチマーク結果は、高水準のスケーラビリティーが容易に達成されることを示しています。 この種のベンチマークの大部分は、カーネル・サービスをほとんど使用しない、小さな CPU 集中プログラムを組み合わせて実行することにより作成されます。 これらのベンチマーク結果は、スケーラビリティーの現実的な予測ではなく、上限を表しています。

ベンチマークについて、もう 1 つ特筆すべき興味深い点は、一般に、1-way SMP は、オペレーティング・システムの UP バージョンを実行する同等のユニプロセッサーより実行が遅い (約 5 から 15%) ことです。

マルチプロセッサーの応答時間

マルチプロセッサーは、個々のプログラムがマルチスレッド化モードで実行できる場合に限り、そのプログラムの実行時間を改善することができます。

単一プログラムの構成部分の並列実行を達成するには、幾つかの方法があります。

- 明示的に libpthreads.a サブルーチン (古いプログラムの場合は **fork()** サブルーチン) に対する呼び出しを行って、同時に実行されるマルチスレッドを作成する。
- 同時に実行できるコードのシーケンスを検出し、それらを並列で実行するためのマルチスレッドを生成する、並列化コンパイラーまたはプリプロセッサーによってプログラムを処理する。
- それ自体がマルチスレッド化されているソフトウェア・パッケージを使用する。

上記の 1 つ以上の手法を使用しない限り、マルチプロセッサー内のプログラムが、同等のユニプロセッサー内のプログラムより高速で実行されることはありません。実際、異なる時点で別々のプロセッサーにディスパッチされるために、より多くのロック・オーバーヘッドおよび遅延が発生することから、プログラムの実行は遅くなる場合があります。

たとえ利用可能な手法をすべて活用したとしても、最大の改善は Amdahl の法則と呼ばれる規則によって限定されます。

- プログラムのユニプロセッサー実行時間 *t* の部分 *x* を順次にしか処理できない場合は、プロセッサーが n 台のマルチプロセッサーにおける実行時間が、同等のユニプロセッサーにおける実行時間よりどれほど向上するか (スピードアップ) は、次の式によって与えられます。

$$speed\ up = \frac{ユニプロセッサー時間}{seq\ 時間 + mp\ 時間} = \frac{t}{xt + \frac{(x-1)t}{n}} = \frac{1}{x + \frac{x}{n}}$$

$$\lim_{n \to \infty} speed\text{-}up = \frac{1}{x}$$

図 *14. Amdahl* の法則： Amdahl の法則によれば、ユニプロセッサーの時間をシーケンス時間とマルチプロセッサー時間の和で割った結果、すなわち 1/(x + (x/n)) が、スピードアップに相当します。 スピードアップの限界は 1/x (n は無限大) です。

例として、プログラムの処理の 50% は順次に行わなければならず、50% は並列に実行できるとすると、最大の応答時間の改善は、2 倍より小 (ほかには何も実行していない 4-way マルチプロセッサーでは、よくて 1.6 倍) です。

SMP スレッドのスケジューリング

SMP 環境では、スレッド・サポートの採用により、SMP を活用するアプリケーションをより容易にかつ費用をかけずにインプリメントすることができます。

スレッド・サポートは、プログラム実行の制御を以下の 2 つのエレメントに分けます。

- メモリーやファイルへのアクセスなど、プログラムの実行に必要な物理リソースの集合であるプロセス。
- 命令アドレス・レジスターや汎用レジスターの現在の内容など、プログラムのインスタンスの実行状態を示すスレッド。 各スレッドは所定のプロセスのコンテキスト内で実行され、そのプロセスのリソースを使用します。 単一プロセス内で複数のスレッドが実行でき、スレッドはプロセスのリソースを共用します。

複数の制御のフローを作成するためにマルチプロセスを fork するのは、煩わしく、費用がかかります。これは、各プロセスがメモリー・リソースの独自のセットを持っていて、セットアップのために多量のシステムの処理を必要とするからです。 単一プロセス内にマルチスレッドを作成する場合は、必要な処理の量も使用するメモリーも少なくてすみます。

スレッド・サポートは、次の 2 つのレベルで存在します。

- アプリケーション・プログラム環境における libpthreads.a サポート
- カーネル・スレッド・サポート

通常、スレッドはマルチプロセッシングを活用するための便利で効率的なメカニズムですが、スレッドにはスケーラビリティーに関する制限があります。 スレッドは処理リソースおよび状態を共有するため、それらのリソースのロックとシリアライゼーションがスケーラビリティーの制限となることがあります。

移行済みワークロードのデフォルト・スケジューラー処理

プロセスとスレッドの間の分割は、既存のプログラムからは隠されています。

事実、以前のリリースのオペレーティング・システムから直接マイグレーションされたワークロードは、いつも行っていたとおりにプロセスを作成します。 各新規プロセスは、他のプロセスのスレッドと CPU を求めて競合する単一スレッド (初期スレッド) と一緒に作成されます。

初期スレッドのデフォルト属性は、新しいスケジューラー・アルゴリズムとともに、未変更のワークロードに対するシステム・ダイナミックスの変更を最小にします。

優先順位は、**nice** コマンドと **renice** コマンド、および **setpri()** システム・コールと **setpriority()** システム・コールによって、従来どおり取り扱うことができます。スケジューラーは、所定のスレッドに、優先順位が同じかそれより高い、次にディスパッチ可能なスレッドに制御を渡すように強制する前に、そのスレッドに 1 タイム・スライス (通常 10 ミリ秒) 間だけ実行を許可します。 詳しくは、 132 ページの『マイクロプロセッサーのコンテンションの制御』を参照してください。

アルゴリズム変数のスケジューリング

幾つかの変数がスレッドのスケジューリングに影響を及ぼします。

一部はスレッド・サポートに固有のもので、その他は、次のようなプロセスのスケジューリング上の考慮事項の詳細です。

優先順位
: スレッドの優先順位の値は、プロセッサー時間のコンテンションにおいて、その優先順位を示す基本インディケーターです。

スケジューラー実行キューの位置
: スケジューラーのディスパッチ可能スレッドのキューにおけるスレッドの位置は、先行する多くの条件に左右されます。

スケジューリング・ポリシー
: このスレッド属性は、タイム・スライスの終わりで実行中のスレッドに何が起こるかを決定します。

コンテンション有効範囲
: スレッドのコンテンション有効範囲は、スレッドがそのプロセス内の他のスレッドとのみ競合するか、またはシステム内のすべてのスレッドと競合するかを決定します。 プロセスのコンテンション有効範囲により作成された pthread はライブラリーによってスケジュールされ、システム有効範囲により作成された pthread はカーネルによってスケジュールされます。 ライブラリー・スケ

ジューラーは、プロセス有効範囲によって pthread をスケジュールするように、カーネル・スレッドのプールを使用します。一般に、pthread が入出力を行う場合は、システム有効範囲によって pthread を作成してください。プロセス有効範囲は、多数のプロセス内同期がある場合に役に立ちます。コンテンション有効範囲は `libpthreads.a` 概念です。

プロセッサーとの親和性
: 親和性が実現される度合いは、パフォーマンスに影響を及ぼします。

これらの考慮事項の組み合わせは複雑に見えるかもしれませんが、所定のプロセスを管理する際には、次の 3 つの異なる方法から選択することができます。

デフォルト
: このプロセスは 1 つのスレッドを持っており、その優先順位は CPU 使用量によって変わり、スケジューリング・ポリシーは SCHED_OTHER です。

プロセス・レベル制御
: プロセスは 1 つ以上のスレッドを持つことができますが、それらのスレッドのスケジューリング・ポリシーはデフォルトの SCHED_OTHER のままです。このポリシーでは、nice の値および固定優先順位を制御する既存の方法を使用することができます。 これらの方法はすべて、プロセス内のすべてのスレッドに等しく影響を及ぼします。 **setpri()** サブルーチンを使用すると、プロセス内のすべてのスレッドのスケジューリング・ポリシーは SCHED_RR に設定されます。

スレッド・レベル制御
: プロセスは 1 つ以上のスレッドを持つことができます。 これらのスレッドのスケジューリング・ポリシーは、SCHED_RR または SCHED_FIFO*n* のいずれか適切な方に設定されます。 各スレッドの優先順位は固定で、スレッド・レベルのサブルーチンによって操作されます。

スケジューリング・ポリシーについては、 46 ページの『スレッドのスケジューリング・ポリシー』を参照してください。

スレッドのチューニング

ユーザー・スレッドは、プロセス内に独立した制御のフローを備えています。

カーネル・サービス (システム・コールなど) にアクセスする必要がある場合、ユーザー・スレッドは、関連するカーネル・スレッドのサービスを受けます。ユーザー・スレッドは、最も注目に値する存在である pthread 共用ライブラリー (`libpthreads.a`) とともに、さまざまなソフトウェア・パッケージで提供されます。 libpthread のインプリメンテーションにより、ユーザー・スレッドは仮想プロセッサー (VP) の上位に置かれ、仮想プロセッサー自体はカーネル・スレッドの上位にあります。 マルチスレッド化ユーザー・プロセスは、次の 2 つのモデルのいずれかを使用することができます。

1:1 スレッド・モデル
: 1:1 モデルは、各ユーザー・スレッドに対して正確に 1 つのカーネル・スレッドをマップすることを表します。 このモデルは、すべての AIX バージョンでデフォルト・モデルです。このモデルで、各ユーザー・スレッドは VP にバインドされ、正確に 1 つのカーネル・スレッドにリンクされます。 VP は必ずしも実際の CPU にバインドする必要はありません (プロセッサーへのバインディングが行われたのでなければ)。 VP にバインドされているスレッドは、システム有効範囲を持っているといわれます。これは、そのスレッドが他のすべてのユーザー・スレッドとともに、カーネル・スケジューラーによって直接スケジュールされるからです。

M:N スレッド・モデル
: M:N モデルは、AIX 4.3.1 でインプリメントされ、現在のデフォルト・モデルにもなっています。このモデルでは、幾つかのユーザー・スレッドが同じ仮想プロセッサーまたは同じ VP のプールを

共用することができます。 各 VP は、ユーザー・コードおよびシステム・コールを実行する際に使用可能な仮想 CPU と考えることができます。 VP にバインドされないスレッドは、ローカルまたはプロセス有効範囲を持っているといわれます。これは、そのスレッドが他のすべてのスレッドとともに、カーネル・スケジューラーによって直接スケジュールされないからです。 pthread ライブラリーは、VP に対するユーザー・スレッドのスケジューリングを処理し、その後カーネルが関連するカーネル・スレッドをスケジュールします。 AIX 4.3.2 では、1 つのカーネル・スレッドを 8 つのユーザー・スレッドにマップするというのがデフォルトです。これは、アプリケーション内から、または環境変数を通じて調整することができます。

アドミニストレーターは、アプリケーションのタイプに応じて、別々のスレッド・モデルを使用することができます。 テストでは、特定のアプリケーションが 1:1 モデルを使用すると、より高いパフォーマンスを発揮できることがわかりました。デフォルトのスレッド・モデルは、AIX 6.1 で M:N から 1:1 に戻されました。すべての AIX バージョンで、特定のプロセスについて環境変数 **AIXTHREAD_SCOPE=S** を設定するだけで、スレッド・モデルを 1:1 に設定し、その後、そのパフォーマンスを、スレッド・モデルが以前 M:N だったときのパフォーマンスと比較することができます。

スレッドを作成し、削除するアプリケーションを見ると、ユーザー・スレッド対カーネル・スレッドのデフォルトの比率が 8:1 であるため、カーネル・スレッドが収穫 *(harvested)* されている場合があります。 この収穫 (harvesting) は、ライブラリー・スケジューリングのオーバーヘッドとともに、パフォーマンスに影響を及ぼすことがあります。これに対して、何千ものユーザー・スレッドが存在する場合は、何千ものカーネル・スレッドを管理するより、ライブラリー内のユーザー・スペースにあるそれらのユーザー・スレッドをスケジュールした方が、オーバーヘッドが少なくてすみます。 pthread の使用中にパフォーマンス上の問題が発生した場合は、常に有効範囲を変更するようにしてください。多くの場合、システム有効範囲によりパフォーマンスを向上させることができます。

SMP システムでアプリケーションを実行中で、ユーザー・スレッドが mutex を獲得できない場合、ユーザー・スレッドは最大 40 回のスピンを試みます。 mutex が短時間のうちに使用可能になる場合もあるので、ある程度長い時間スピンするというのも、時間を費やすだけのことはあるはずです。 CPU を追加するにつれてパフォーマンスが低下する場合、これは通常ロックの問題を示しています。 環境変数を **SPINLOOPTIME=***n* に設定してスピン時間を増やすことも考えられます。ここで *n* はスピンの数です。CPU の速度と CPU の数に応じて、この値を数千という高い値に設定することは、異常ではありません。スピン・カウントを使い切ったら、スレッドはスリープ状態に入って mutex が使用可能になるのを待つことも、**yield()** システム・コールを出して単に CPU の制御を渡した上で、スリープ状態には入らず、実行可能状態にとどまることもできます。 デフォルトでは、スレッドはスリープ状態に入りますが、**YIELDLOOPTIME** 環境変数をある数に設定すると、その設定した回数まで制御を手放してからスリープ状態に入ります。 渡した後で CPU を入手するたびに、スレッドは mutex の獲得を試みることができます。

malloc サブシステムを多用する特定のマルチスレッド化ユーザー・プロセスは、アプリケーションを始動する前に環境変数 **MALLOCMULTIHEAP=1** をエクスポートすることにより、パフォーマンスを改善できる場合があります。パフォーマンスが改善される可能性が特に高いのは、マルチスレッド化 C++ プログラムの場合です。その理由は、これらのプログラムが、コンストラクターまたはデストラクターが呼び出されると必ず malloc サブシステムを使用するからです。 有効なパフォーマンスの改善が最も顕著に認められるのは、マルチスレッド化ユーザー・プロセスを SMP システムで実行中の場合で、特にシステム有効範囲スレッドが使用されている場合です (M:N の比率が 1:1)。 ただし、場合によっては、その他の条件下でも、またユニプロセッサー上でも、明らかな改善が見られることがあります。

スレッド環境変数

libpthreads.a フレームワーク内には、アプリケーションのパフォーマンスに影響する可能性がある、一連のチューニング・ノブが提供されています。

可能であれば、バイナリー実行可能プログラムを起動するためのフロントエンド・シェル・スクリプトを使用します。 シェル・スクリプトには、環境変数に対してこの後のセクションに記述されているシステム・デフォルト値をオーバーライドするために、新規の値を指定してください。

AIXTHREAD_COND_DEBUG

AIXTHREAD_COND_DEBUG 変数は、デバッガーが使用する条件変数のリストを維持します。 プログラムに多数のアクティブな条件変数が含まれていて、条件変数の作成と破棄を頻繁に行う場合、これによって、条件変数のリストを維持するためのより多くのオーバーヘッドが発生する可能性があります。この変数を **OFF** に設定すると、リストが使用不可になります。 この変数をオンのままにすれば、スレッド化されたアプリケーションのデバッグが容易になりますが、いくらかのオーバーヘッドが発生します。

AIXTHREAD_ENRUSG

AIXTHREAD_ENRUSG 変数は、pthread リソース収集を使用可能または使用不可にします。 オンにすると、プロセス内の pthreads すべてのリソース収集が可能になりますが、オーバーヘッドがかかることになります。

AIXTHREAD_GUARDPAGES=n

```
*      +-----------------------+
*      | pthread attr          |
*      +-----------------------+ <--- pthread->pt_attr
*      | pthread struct        |
*      +-----------------------+ <--- pthread->pt_stk.st_limit
*      | pthread stack         |
*      |   |                   |
*      |   V                   |
*      +-----------------------+ <--- pthread->pt_stk.st_base
*      | RED ZONE              |
*      +-----------------------+ <--- pthread->pt_guardaddr
*      | pthread private data  |
*      +-----------------------+ <--- pthread->pt_data
```

上図の RED ZONE は *Guardpage* と呼ばれています。

pthread attr、pthread、および ctx は、pthread に割り当てられたメモリーの PTH_FIXED 部分を表します。

下図のおおよそのバイト・サイズは、32 ビットでは大括弧の中に示されています。 64 ビットの場合、PTH_FIXED を構成する部分は多少大きく、キー・データは 8 K と予想されますが、それ以外は同じです。

```
*      +-----------------------+
*      | page alignment 2      |
*      | [8K-4K+PTH_FIXED-a1]  |
*      +-----------------------+
*      | pthread ctx [368]     |
*      +-----------------------+<--- pthread->pt_attr
*      | pthread attr [112]    |
*      +-----------------------+ <--- pthread->pt_attr
*      | pthread struct [960]  |
*      +-----------------------+ <--- pthread
*      | pthread stack         |         pthread->pt_stk.st_limit
*      |   |[96K+4K-PTH_FIXED] |
```

```
*      |   V                    |
*      +------------------------+ <--- pthread->pt_stk.st_base
*      | RED ZONE [4K]          |
*      +------------------------+ <--- pthread->pt_guardaddr
*      | pthread key data [4K]  |
*      +------------------------+ <--- pthread->pt_data
*      | page alignment 1 (a1)  |
*      | [<4K]                  |
*      +------------------------+
```

上図の RED ZONE は Guardpage と呼ばれています。

pthread スタックの終わりに追加する 10 進数の保護ページ数は *n* です。この値は pthread の作成時に指定される属性値をオーバーライドします。アプリケーションが独自のスタックを指定する場合は、保護ページは作成されません。デフォルトは 0 で、*n* は正の値でなければなりません。

保護ページのバイト数でのサイズは、*n* に PAGESIZE を掛けて求めます。 ページ・サイズは、システムによって決められるサイズです。

AIXTHREAD_DISCLAIM_GUARDPAGES

AIXTHREAD_DISCLAIM_GUARDPAGES 変数は、pthread スタックが作成されるときにスタック保護ページが放棄されるかどうかを制御します。 **AIXTHREAD_DISCLAIM_GUARDPAGES=ON** の場合、保護ページは放棄されます。 pthread スタックに保護ページがない場合、**AIXTHREAD_DISCLAIM_GUARDPAGES** 変数を設定しても無効です。

AIXTHREAD_MNRATIO

AIXTHREAD_MNRATIO 変数は、ライブラリーのスケール因数を制御します。 この比率は、pthread の作成および終了時に使用されます。 非常に多数のスレッドを使用するアプリケーションに役立ちます。 ただし、パフォーマンスを改善できるので、常に 1:1 の比率をテストしてください。

AIXTHREAD_MUTEX_DEBUG

AIXTHREAD_MUTEX_DEBUG 変数は、デバッガーが使用するアクティブな mutex のリストを維持します。 プログラムに多数のアクティブな mutex が含まれていて、mutex の作成と破棄を頻繁に行う場合、これによって mutex のリストを保持するためのより多くのオーバーヘッドが発生する可能性があります。この変数を ON に設定すると、スレッド化されたアプリケーションのデバッグが容易になりますが、オーバーヘッドが増えます。この変数を OFF 設定のままにしておくと、リストは使用不可です。

AIXTHREAD_MUTEX_FAST

プログラムは過度の mutex コンテンションのためにパフォーマンスが低下する場合は、この変数を ON に設定して pthread ライブラリーがプロセス専用 mutex のみで作動するように、最適化した mutex のロック・メカニズムを必ず使用します。 これらのプロセス専用 mutex は pthread_mutex_init ルーチンを使用して初期化し、破壊するときは pthread_mutex_destroy ルーチンを使用する必要があります。 この変数を OFF 設定のままにすると、pthread ライブラリーはデフォルト mutex のロック・メカニズムを使用します。

AIXTHREAD_READ_GUARDPAGES

AIXTHREAD_READ_GUARDPAGES 変数は、pthread スタックの終わりに追加される保護ページに対する読み取りアクセスを使用可能または使用不可にします。 pthread によって作成される保護ページについて詳しくは、 78 ページの『AIXTHREAD_GUARDPAGES=n』を参照してください。

AIXTHREAD_RWLOCK_DEBUG

AIXTHREAD_RWLOCK_DEBUG 変数は、デバッガーが使用する読み取り/書き込みロックのリストを保守します。 プログラムに多数のアクティブな読み取り/書き込みロックが含まれていて、読み取り/書き込みロックの作成と破棄を頻繁に行う場合、これによって、読み取り/書き込みロックのリストを保持するためのより多くのオーバーヘッドが発生する可能性があります。 この変数を `OFF` に設定すると、リストが使用不可になります。

AIXTHREAD_SUSPENDIBLE={ON|OFF}

AIXTHREAD_SUSPENDIBLE 変数を `ON` に設定すると、**pthread_suspend_np** ルーチンまたは **pthread_suspend_others_np** ルーチンで以下のルーチンを使用するアプリケーションのデッドロックを防ぎます。

- pthread_getrusage_np
- pthread_cancel
- pthread_detach
- pthread_join
- pthread_getunique_np
- pthread_join_np
- pthread_setschedparam
- pthread_getschedparam
- pthread_kill

この変数に関連して、パフォーマンスが少し低下します。

AIXTHREAD_SCOPE={S|P}

S オプションはシステム 全体のコンテンションの有効範囲 (1:1) を意味し、**P** オプションはプロセス 全体のコンテンションの有効範囲 (M:N) を意味します。これらのオプションのどちらかを指定する必要があります。デフォルト値は **S** です。

AIXTHREAD_SCOPE 環境変数を使用すると、デフォルト属性を用いて作成されたスレッドにのみ影響を与えます。デフォルト属性が使用されるのは、`pthread_create()` サブルーチンに対する *attr* パラメーターが NULL の場合です。

ユーザー・スレッドがシステム全体の有効範囲を指定して作成される場合、そのユーザー・スレッドはカーネル・スレッドにバインドされ、カーネルによってスケジュールされます。ベースとなるカーネル・スレッドは、他のユーザー・スレッドとは共用されません。

ユーザー・スレッドがプロセス全体の有効範囲を指定して作成される場合、そのユーザー・スレッドはユーザー・スケジューラーに制御されます。つまり、以下のようになります。

- 専用のカーネル・スレッドがない。
- ユーザー・モードでスリープする。
- プロセッサーを待つ間ユーザー実行キューに置かれる。
- ユーザー・スケジューラーによるタイム・スライシングの対象になる。

テストでは、一部のアプリケーションが 1:1 モデルでより高いパフォーマンスを発揮できることがわかりました。

AIXTHREAD_SLPRATIO

AIXTHREAD_SLPRATIO スレッド・チューニング変数は、スリープ中のスレッド用に予約する必要があるカーネル・スレッドの数を制御します。 一般的に、スリープ中の pthread は通常、一度に 1 つずつウェイクされるので、サポートするために必要なカーネル・スレッドはより少なくなります。 これにより、カーネル・リソースが節約できます。

AIXTHREAD_STK=n

AIXTHREAD_STK=n スレッド・チューニング変数は、それぞれの pthread に割り振るべきバイト数 (10 進数) を制御します。 この値は、**pthread_attr_setstacksize** によってオーバーライドすることができます。

AIXTHREAD_AFFINITY={default|strict|first-touch}

AIXTHREAD_AFFINITY は、pthread 構造、スタック、およびスレッド・ローカル・ストレージの配置を、拡張アフィニティー使用可能化システムで制御します。

- 「default」オプションはこのデータの特別な配置を行おうとはしません。この場合、システム設定で決定されたように、プロセスが使用するメモリー領域全体にわたって配置をバランスさせます。
- 「strict」オプションは、必ず、このデータを pthread のローカル・メモリー内に配置します。すなわち、これによって pthread の作成時にパフォーマンス的に不利な条件が若干発生する可能性があります。その理由は、あるメモリー領域から別メモリー領域に既存データがマイグレーションされるからです。しかし、実行時のパフォーマンスを向上することができます。
- 「first-touch」オプションは、pthread のローカル・メモリー内に配置することにおいては類似していますが、メモリー内でのデータのマイグレーションを行おうとはしません。 メモリー内ページは、スレッドにより、このデータに対して必要とされる (ページング・スペースからのメモリー内へのページングを含む) と、ローカルに配置されます。 このオプションを使用すると、起動時と実行時のパフォーマンスの間でバランスを取ることができます。

AIXTHREAD_PREALLOC=n

AIXTHREAD_PREALLOC 変数は、スレッド作成時に事前割り振りと解放を行うバイト数を指定します。一部のマルチスレッド化されたアプリケーションは、複数のスレッドから同時に sbrk() が呼び出されないようにすることで、この利点を活用できる場合があります。

デフォルトは 0 で、*n* は正の値でなければなりません。

AIXTHREAD_HRT

AIXTHREAD_HRT=true 変数は、アプリケーションの `pthreads` に対する高解像度のタイムアウトを可能にします。高解像度のタイムアウトを使用可能にするには、`root` 権限または `CAP_NUMA_ATTACH` 権限を持っている必要があります。 これらの必要な権限または能力を持っていない場合、この環境変数は無視されます。

MALLOCBUCKETS

Malloc バケットは、オプションのバケット・ベースの拡張デフォルト・アロケーターを提供します。 これは、大量の小さな割り当て要求を出すアプリケーションのために、malloc のパフォーマンスを改善するためのものです。 malloc バケットが使用可能になっている場合、事前定義されたブロック・サイズの範囲に入る割り当て要求は、malloc バケットによって処理されます。 その他のすべての要求は、デフォルトのアロケーターによって通常の方法で処理されます。

Malloc バケットは、デフォルトでは使用可能になりません。 プロセスの起動に先立って、*MALLOCTYPE* および *MALLOCBUCKETS* 環境変数を設定することによって、使用可能にされ、設定されます。

malloc バケットについて詳しくは、プログラミングの一般概念: プログラムの作成およびデバッグを参照してください。

MALLOCMULTIHEAP={considersize,heaps:n}

スレッド化アプリケーションが **malloc()**、**free()**、および **realloc()** サブルーチン呼び出しを出して複数のスレッドを持つことができるように、複数のヒープが必要です。 単一ヒープの場合、**malloc()**、**free()**、または **realloc()** 呼び出しを試行するスレッドはすべてシリアライズされます (呼び出しは一度に 1 回だけです)。 その結果はマルチプロセッサー・マシンに重大な影響を与えます。複数のヒープを使用すると、各スレッドは独自のヒープを入手します。 すべてのヒープが使用されている場合、呼び出しを試行する新しいスレッドはどれも、1 つ以上のヒープが再び使用可能になるまで待機する必要があります。 なお、シリアライズが発生する場合でも、その発生の可能性と発生したときの影響は非常に軽減されます。

スレッド・セーフ・ロックは、このアプローチを取り扱うように変更されています。 各ヒープは専用のロックを持ち、ロック・ルーチンはシリアライゼーションを防止しようと「インテリジェントに」ヒープを選択します。 *MALLOCMULTIHEAP* 環境変数に **considersize** が設定されると、その選択により、さらに、単に次のアンロック状態のヒープを選択するのではなく、要求を処理するための十分なフリー・スペースを持つ、使用可能なヒープを選択しようとします。

複数のオプションを指定する場合は、次の例のように、(任意の順序で) コンマで区切ります。

```
MALLOCMULTIHEAP=considersize,heaps:3
```

以下のオプションがあります。

considersize
: プロセスの作業セット・サイズを最小化しようとする、別のヒープ選択アルゴリズムを使用します。 デフォルトはこのオプションを使用しないで、より高速のアルゴリズムを使用します。

heaps:*n*
: ヒープ数を変更する場合は、このオプションを使用します。 *n* の有効な範囲は 1 から 32 までです。 *n* にこの範囲外の数 (*n*<=0 または *n*>32) を設定すると、*n* には 32 が設定されます。

MALLOCMULTIHEAP のデフォルトは NOT SET (最初のヒープのみが使用される) です。 環境変数 *MALLOCMULTIHEAP* が設定された場合 (例えば、`MALLOCMULTIHEAP=1`)、スレッド化アプリケーションは 32 のヒープすべてを使用することができます。 `MALLOCMULTIHEAP=heaps:`*n* と設定すると、ヒープの数は 32 ではなく *n* に制限されます。

詳しくは、プログラミングの一般概念: プログラムの作成およびデバッグの『Malloc マルチヒープ』のセクションを参照してください。

SPINLOOPTIME=n

SPINLOOPTIME 変数は、プロセスを発生させるためにカーネルを呼び出すといった 2 次的なアクションをとらないで、システムがビジー mutex またはスピン・ロックの獲得を試みる回数を制御します。 この制御は MP システムを対象としたものです。このシステムでは、別のアクティブに実行中の pthread によって保持されているロックの解放が期待されています。 このパラメーターは、libpthread (ユーザー・スレッド) 内でのみ機能します。 ロックが通常短時間だけ使用可能な場合は、この環境変数を設定してスピン時間を増やすことができます。 別の pthread に制御を渡す前にビジー・ロックを再試行する回数は *n* です。デフォルトは 40 で、*n* は正の値でなければなりません。

MAXSPIN カーネル・パラメーターは、カーネル・ロック・ルーチンにおけるスピンに影響します (86 ページの『MAXSPIN パラメーターの変更のための schedo コマンドの使用法』を参照してください)。

YIELDLOOPTIME=n

YIELDLOOPTIME 変数を使用して、システムがビジー mutex またはスピン・ロックの獲得を試みている間、実際にロック上でスリープする前に、プロセッサーに制御を渡す回数を制御します。 プロセッサーは、十分な優先順位を持つ別の実行可能スレッドが存在すると想定して、別のカーネル・スレッドに渡されます。この変数は、複数ロックを使用中の複合アプリケーションで有効であることが示されています。 ビジー・ロック上でのブロッキングの前にプロセッサーの制御を渡す回数は *n* です。デフォルトは 0 で、*n* は正の値でなければなりません。

プロセス全体のコンテンション有効範囲の変数

以下の環境変数は、プロセス全体のコンテンション有効範囲を指定して作成される pthread のスケジューリングに影響を与えます。

AIXTHREAD_MNRATIO=*p*:*k*

ここで *k* は、*p* 個の実行可能 pthread を処理するために使用すべきカーネル・スレッドの数です。 この環境変数は、ライブラリーのスケール・ファクターを制御します。 この比率は、pthread の作成および終了時に使用されます。 この変数はプロセス全体の有効範囲でのみ有効です。システム全体の有効範囲の場合、この環境変数は無視されます。 デフォルトの設定値は 8:1 です。

AIXTHREAD_SLPRATIO=*k*:*p*

ここで *k* は、*p* 個のスリープ pthread の予備に取っておくべきカーネル・スレッドの数です。このスリープ率は、スリープ pthread をサポートするために余分に保持するカーネル・スレッドの数です。 スリープ pthread は概して一度に 1 つずつ起動されるので、一般に、スリープ pthread のサポートに要するカーネル・スレッドの数は少なくてすみます。 これにより、カーネル・リソースが節約できます。 *p* および *k* には、任意の正整数値を指定することができます。*k*>*p* の場合、その比率は 1:1 として扱われます。 デフォルトは 1:12 です。

AIXTHREAD_MINKTHREADS=*n*

ここで *n* は、使用すべきカーネル・スレッドの最小数です。 ライブラリー・スケジューラーは、この数字を下回ったカーネル・スレッドは再利用しません。カーネル・スレッドは、事実上どの時点でも再利用することができます。 一般に、カーネル・スレッドは、pthread の終了の結果、再利用のターゲットとなります。 デフォルトは 8 です。

スレッド・デバッグ・オプション

pthread ライブラリーは、デバッガーによって使用されるアクティブ mutex のリスト、条件変数、および読み取り/書き込みロックを維持します。

ロックが初期化されると、リストには既にないものと想定され、そのロックはリストに追加されます。 このリストはリンク・リストとして保持されるので、新しいロックが既にリストにはないとの判断は、リストが大きくなったとき、パフォーマンスに影響を与えます。 この問題は、リストがロック (dbx__mutexes) によって保護されているという事実により、さらに複雑になります。このロックはリストの検索全体にわたって保留されるからです。 この場合、検索が行われている間、**pthread_mutex_init()** サブルーチンに対するその他の呼び出しは保留されます。

以下の環境変数が OFF (デフォルト) に設定された場合、該当するデバッグ・リストは完全に使用不可となります。 これは、**dbx** コマンド (または pthread デバッグ・ライブラリーを使用する任意のデバッガー) が、オブジェクトの不在を示すことを意味します。

- **AIXTHREAD_MUTEX_DEBUG**
- **AIXTHREAD_COND_DEBUG**
- **AIXTHREAD_RWLOCK_DEBUG**

これらの環境変数のいずれかを ON に設定するには、以下のコマンドを使用します。

```
# export variable_name=ON
```

SMP ツール

オペレーティング・システムのパフォーマンス・ツールはすべて SMP マシンをサポートします。

一部のパフォーマンス・ツールでは、個々のプロセッサーの使用率統計情報が提供されます。 その他のパフォーマンス・ツールでは、すべてのプロセッサーの使用率統計情報を平均し、その平均だけが表示されます。

このセクションでは、SMP 上でのみサポートされるツールについて説明します。その他のすべてのパフォーマンス・ツールについての詳細は、該当するセクションを参照してください。

bindprocessor コマンド

プロセスのカーネル・スレッドをプロセッサーにバインドまたはアンバインドするときは、**bindprocessor** コマンドを使用します。

所有していないプロセス内のスレッドをバインドまたはアンバインドするには、root 権限が必要です。

注: **bindprocessor** コマンドはマルチプロセッサー・システムを対象としています。 このコマンドはユニプロセッサー・システムでも機能しますが、この種のシステムではバインディングは無効です。

使用可能なプロセッサーを照会するには、以下を実行します。

```
# bindprocessor -q
The available processors are:  0 1 2 3
```

その出力は使用可能なプロセッサーの論理プロセッサー番号を示し、それらは後で示すように、**bindprocessor** コマンドで使用します。

PID が 14596 のプロセスをプロセッサー 1 にバインドするには、以下を実行します。

```
# bindprocessor 14596 1
```

コマンドが正常に実行された場合は、リターン・メッセージは戻されません。 プロセスがプロセッサーにバインドまたはアンバインドされているかどうか検査するには、 124 ページの『ps コマンドの使用法』に説明されているように、**ps -mo THREAD** コマンドを使用します。

```
# ps -mo THREAD
USER   PID  PPID    TID ST  CP PRI SC    WCHAN        F     TT BND COMMAND
root  3292  7130      - A    1  60  1        -   240001  pts/0   - -ksh
   -     -     -  14309 S    1  60  1        -      400      -   - -
root 14596  3292      - A   73 100  1        -   200001  pts/0   1 /tmp/cpubound
   -     -     -  15629 R   73 100  1        -        0      -   1 -
root 15606  3292      - A   74 101  1        -   200001  pts/0   - /tmp/cpubound
   -     -     -  16895 R   74 101  1        -        0      -   - -
root 16634  3292      - A   73 100  1        -   200001  pts/0   - /tmp/cpubound
   -     -     -  15107 R   73 100  1        -        0      -   - -
root 18048  3292      - A   14  67  1        -   200001  pts/0   - ps -mo THREAD
   -     -     -  17801 R   14  67  1        -        0      -   - -
```

「*BND*」欄は、プロセスがバインドされているプロセッサーの数を示し、プロセスがまったくバインドされていない場合はダッシュ (-) が表示されます。

PID が 14596 のプロセスをアンバインドするには、以下のコマンドを使用します。

```
# bindprocessor -u 14596
# ps -mo THREAD
USER   PID  PPID    TID ST  CP PRI SC    WCHAN        F     TT BND COMMAND
root  3292  7130      - A    2  61  1        -   240001  pts/0   - -ksh
   -     -     -  14309 S    2  61  1        -      400      -   - -
root 14596  3292      - A  120 124  1        -   200001  pts/0   - /tmp/cpubound
   -     -     -  15629 R  120 124  1        -        0      -   - -
root 15606  3292      - A  120 124  1        -   200001  pts/0   - /tmp/cpubound
   -     -     -  16895 R  120 124  1        -        0      -   - -
root 16634  3292      - A  120 124  0        -   200001  pts/0   - /tmp/cpubound
   -     -     -  15107 R  120 124  0        -        0      -   - -
root 18052  3292      - A   12  66  1        -   200001  pts/0   - ps -mo THREAD
   -     -     -  17805 R   12  66  1        -        0      -   - -
```

あるプロセスで **bindprocessor** コマンドを使用すると、そのスレッドはすべて 1 つのプロセッサーにバインドされ、以前バインドされていたプロセッサーからアンバインドされます。 プロセスをアンバインドすると、そのスレッドもすべてアンバインドされます。 **bindprocessor** コマンドを使用して、個々のスレッドをバインドまたはアンバインドすることはできません。

ただし、1 つのプログラム内では、**bindprocessor()** 関数呼び出しを使用して個々のスレッドをバインドすることができます。 コードの一部で **bindprocessor()** 関数を使用してスレッドをプロセッサーにバインドすると、そのスレッドはこれらのプロセッサーとともに残り、アンバインドすることはできません。 当該プロセスで **bindprocessor** コマンドを使用すると、そのスレッドはすべて 1 つのプロセッサーにバインドされ、それぞれが以前バインドされていたプロセッサーからアンバインドされます。 プロセス全体をアンバインドすると、そのスレッドもすべてアンバインドされます。

プロセスは、始動されるまではバインドすることができません。つまり、バインドするためには、プロセスが存在していなければなりません。 プロセスが存在しない場合は、以下のエラーが表示されます。

```
# bindprocessor 7359 1
1730-002: Process 7359 does not match an existing process
```

プロセッサーが存在しない場合は、以下のエラーが表示されます。

```
# bindprocessor 7358 4
1730-001: Processor 4 is not available
```

注**:** 待ちプロセス **kproc** に **bindprocessor** コマンドを使用しないでください。

バインディングに関する考慮事項**:**

プロセス・バインディングを使用する前に、幾つかの考慮すべき問題があります。

バインディングは、ほとんど割り込みが起こらない CPU 集中プログラムに役立つ場合もあります。通常のプログラムの場合、バインディングは時には逆効果になります。これは、入出力後、スレッドがバインドされるプロセッサーが使用可能になるまで、スレッドの再ディスパッチを遅らせる可能性があるからです。スレッドが入出力操作の間中ブロックされている場合、その処理コンテキストの多くが、スレッドがバインドされているプロセッサーのキャッシュに残っているとは考えられません。スレッドを、次に使用可能なプロセッサーにディスパッチした方が、より適切な処理が行われます。

バインディングは、ユーザーが自分のプロセスをバインドしたプロセッサーに、その他のプロセスがディスパッチされることを妨げません。 バインディングは区分化とは異なります。 rsets または排他的 rsets を使用すると、一連の論理プロセッサーを特定のワークロード専用にすることができます。したがって、ユ

ーザーが自分のプロセスをバインドしたプロセッサーに、より優先順位の高いプロセスがディスパッチされる可能性があります。この場合、ユーザーのプロセスはその他のプロセッサーにディスパッチされません。そのため、このバインド・プロセスのパフォーマンスは向上しません。バインド・プロセスの優先順位を上げれば、よりよい結果が達成されることもあります。

非常に負荷の大きいシステムでプロセスをバインドした場合は、そのパフォーマンスが低下する可能性があります。これは、プロセッサーがアイドルになったとき、そのプロセッサーがプロセスをバインドしたプロセッサーでない場合は、そのアイドル・プロセッサーでプロセスを実行することはできないからです。

プロセスがマルチスレッド化されている場合、プロセスをバインドすると、そのスレッドもすべて同じプロセッサーにバインドされます。 したがって、プロセスはマルチプロセッシングの利点を生かすことができず、パフォーマンスは向上しません。

注**:** プロセス・バインディングは、AIX によって提供される本来のロード・バランシングを妨げ、システム全体のパフォーマンスを低下させることがあるため、注意して使用してください。システムのワークロードが、モニター対象である初期バインディングから変更されている場合は、システム・パフォーマンスが損なわれることがあります。 **bindprocessor** コマンドを使用する場合は、環境が変更され、バインド・プロセスがシステム・パフォーマンスを低下させる可能性があるので、定期的にシステムをモニターしてください。

MAXSPIN パラメーターの変更のための schedo コマンドの使用法

あるスレッドが、現在別のスレッドが所有し、別の CPU で実行中のロックの獲得を必要とする場合、ロックを必要とするスレッドは、オーナー・スレッドがロックを解放するまでの間、*MAXSPIN* というチューナブル・パラメーターで指定されている特定の値の回数まで、CPU 上でスピンします。

MAXSPIN のデフォルト値は、SMP システムでは 0x4000 (16384)、UP システムでは 1 です。 あるシステムで、以前は示されなかったアイドル時間または入出力待ち時間の増加に気付いた場合は、スレッドがたびたびスリープ状態になっている可能性があります。 これによってパフォーマンス上の問題が生じる場合は、*MAXSPIN* をもっと大きい値にチューニングするか、-1 (0xFFFFFFFF 回までスピンすることを意味する) に設定してください。

スリープ状態に入るまでのスピン回数を変更するには、**schedo** コマンドの **maxspin** オプションを使用します。 過剰なスピンによって生じる可能性がある CPU 使用率を低減するには、次のように入力して、*MAXSPIN* の値を減らしてください。

```
# schedo -o maxspin=8192
```

その結果、コンテキスト切り替えの増加が認められることがあります。 コンテキスト切り替えがボトルネックになる場合は、*MAXSPIN* を増やしてください。

値を変更するには、root ユーザーになる必要があります。

パフォーマンスの計画とインプリメンテーション

プログラムのパフォーマンスが条件に合わない場合、それは機能的とはいえません。すべてのプログラムは、ユーザーの集まり、ときには大規模で多様なユーザーの集まりを満足させる必要があります。 プログラムのパフォーマンスが、これらのユーザーの相当数に本当に受け入れられなければ、そのプログラムは使用されません。 プログラムが使用されなければ、その意図した機能は実行されません。

この状況は、ユーザー作成のアプリケーションだけではなく、ライセンス交付を受けたソフトウェア・パッケージでも同じく当てはまります。ソフトウェア・パッケージの開発者の多くは、低パフォーマンスの影響

に気付き、プログラムをできる限り高速で実行できるように骨を折っているのですが。 残念ながら、開発者は、そのプログラムが使用される環境や使用法のすべてを予想することはできません。 パフォーマンスが受け入れられるかどうかの最終的な責任は、ソフトウェア・パッケージを選択または作成し、計画し、インストールする人にあります。

このセクションでは、プログラマーまたはシステム管理者が、新たに作成または購入したプログラムのパフォーマンスを許容できるものにするための過程について説明します。 (プログラマー という語が単独で使われている場合、この用語は常に、システム管理者およびプログラムの最終的な成功に責任がある他のすべての担当者を含みます。)

プログラムで許容できるパフォーマンスを達成するためには、プロジェクトの開始時に許容可能度の確認および定量化を行い、それを達成するのに必要な測定基準およびリソースを見失わないようにしてください。これは基本的なやり方のように思われますが、一部のプログラミング・プロジェクトでは、意識的にこれを無視しています。これらのプロジェクトでは、設計、コーディング、デバッグ、場合によっては文書化といったポリシーを採用し、そして時間があればパフォーマンスの改良を行う程度です。

プログラムを予想どおり、単にロジックにおいてだけでなく、一定の時間で機能するように作成する唯一の方法は、パフォーマンス上の考慮をソフトウェアの計画および開発過程に統合することです。 既存のソフトウェアをインストールする場合は、事前の計画がさらに重要になることもあります。インストールの担当者には開発者ほど自由がないからです。

このプロセスの詳細は、小さいプログラムの場合は煩わしくても、2 番目の「課題」があることを覚えておいてください。 新しいプログラムのパフォーマンスが満足のいくものでなければならないのはもちろん、さらにそのプログラムを既存のシステムに追加した場合に、そのシステム上で実行されるその他のプログラムのパフォーマンスが決して低下しないようにする必要があります。

ワークロードのコンポーネント識別

プログラムが新たに作成したものであれ購入したものであれ、あるいは小さくても大きくても、開発者、インストール担当者、および将来のユーザーはそれぞれ、プログラムの使用について前提事項を検討する必要があります。

以下の前提事項が必要になります。

- 誰がプログラムを使用するか
- プログラムが実行される状態
- そのような状態が生じる頻度、およびそれは何年、何月、何日の何時であるか
- それらの状態が既存のプログラムの追加使用も必要とするか
- そのプログラムをどのシステムで実行するか
- 処理されるデータの量、およびどこからのデータを処理するのか
- そのプログラムによって、またはそのプログラムのために作成されたデータを、その他の方法で使用するか

これらのアイデアは、設計プロセスの一部として顕在化させない限り、多くの場合漠然としており、プログラマーはほぼ間違いなく、将来のユーザーとは異なる前提事項を持つことになります。 プログラマーがユーザーでもあるという場合でも、前提事項をあいまいなままにしておくと、設計と前提事項とを厳密な方法で比較することは不可能になります。なお悪いことに、実行される作業についての完全な理解なくしては、パフォーマンス要件を識別することは不可能です。

パフォーマンス要件文書

パフォーマンス要件の確認および定量化においては、特定の要件の背後にある理論を確認することが重要です。 これは、一般のキャパシティー・プランニング・プロセスの一部です。 ユーザーは、プログラムのロジックにかかわる前提事項に関して、プログラマーの前提事項とは一致しない要件に基づいて主張するかもしれません。

パフォーマンス要件には最小限、以下の事項を文書化しておく必要があります。

- ユーザー・コンピューター間対話の各タイプごとに、大部分の時間帯で満足のいく最大の応答時間 (大部分の時間帯 の定義とともに)。 応答時間は、ユーザーが「開始」というアクションをとった時点から、ユーザーがタスクを継続するのに十分なフィードバックをコンピューターから受け取るまでの時間で測定されます。 応答時間は、ユーザーが感じる待ち時間です。サブルーチンへの入り口から最初の書き込みステートメントまでの時間という意味ではありません。

  ユーザーが応答時間に関心がなく、結果だけが関心事であるという場合、プログラマーは、「現在のスタンドアロンでの実行時間の見積もりの 10 倍」が受け入れ可能かどうか尋ねることができます。 その応答が「はい」であれば、プログラマーはスループットのディスカッションに進むことができます。 「はい」でない場合は、ユーザーの十分な注目を集めて、応答時間に関するディスカッションを続けます。
- その他の時間帯に最小限許容できる応答時間。 応答時間が長いと、ユーザーはシステムが停止していると考える場合があります。 プログラマーはその他の時間帯 も指定する必要があります。例えば、1 日のうちのピーク (分) を、対話全体の 1% にします。 応答時間の低下が、1 日の特定の時間に集中すると影響が大きくなります。
- 必要とされる通常のスループットおよび発生する時間帯。これは単なる思いつきの考慮事項ではありません。 例えば、あるプログラムの要件が、午前 10 時および午後 3 時 15 分の 1 日 2 回実行することだとします。これが 15 分間実行される CPU 制約のプログラムで、マルチユーザー・システムで実行する予定の場合は、多少のネゴシエーションが必要です。
- 最大スループット期間のサイズおよびタイミング。
- 予想される各種要求の混成、およびその混成が時間の経過とともにどのように変化するか。
- マシン当たりのユーザー数、およびこれがマルチユーザー・アプリケーションの場合は、ユーザーの合計数。 この記述には、これらのユーザーがログオンおよびログオフする時間帯に加えて、そのキー・ストロークの想定速度、完了要求、および考慮時間も入れる必要があります。 考慮時間が先行する要求と後続の要求によって規則正しく変化するかどうかを調べることも必要です。
- ワークロードが実行されるマシンについて、ユーザーが想定するすべての事項。 ユーザーが既存の特定マシンを念頭に置いている場合、プログラマーはそのことをあらかじめ承知しておく必要があります。 同様に、ユーザーが特定のタイプ、サイズ、コスト、ロケーション、相互接続、その他、前述の要件を満足するプログラマーの能力を制約するような、何らかの変数を想定している場合は、その前提事項も要件の一部となります。 プログラムが開発され、テストされ、または初めてインストールされるシステムでは、多くの場合満足度は評価されません。

ワークロードのリソース要件見積もり

詳細なリソース要件のドキュメンテーションが一緒に提供されるソフトウェア・パッケージを購入するのでなければ、リソースの見積もりは、パフォーマンス計画プロセスにおいて最も困難な作業になります。

この障害には、次のような幾つかの原因があります。

- どのタスクを行うにも、幾つかの方法があります。 C (またはその他の高水準言語) プログラム、シェル・スクリプト、**perl** スクリプト、**awk** スクリプト、**sed** スクリプト、AIX ウィンドウ・ダイアログ

などを作成することができます。 一部の手法はアルゴリズムやプログラマーの生産性に特に適しているように見えても、パフォーマンスの観点からは、並外れて費用がかかります。

役に立つガイドラインとしては、抽象化のレベルが高ければ高いほど、パフォーマンスが予期しない結果にならないように十分な注意が必要だということです。 一見無害な構成に含まれるデータ・ボリュームや反復の数を、注意深く検討してください。

- 単一プロセスの正確なコストを定義するのは困難です。 この障害は、単に技術的なだけでなく、哲学的なものです。 複数のユーザーによって実行される特定のプログラムの複数インスタンスがプログラム・テキストのページを共用している場合、それらのメモリーのページのコストはどのプロセスが負担すべきでしょうか。 オペレーティング・システムは、最近使用されたファイル・ページをメモリー内に残して、そのデータに再度アクセスするプログラムにキャッシング効果を与えます。 データに再度アクセスするプログラムは、そのデータを保存するために使用されたスペースのコストを負担すべきでしょうか。 システム・クロックなどのある種の測定の細分性により、同じプログラムの連続するインスタンスによるものとされる CPU 時間に変動が生じることがあります。

  リソース・レポートのあいまいさと変動性に対処する 2 つの方法があります。 1 番目の方法は、あいまいさを無視し、測定値の整合性が許容できる程度になるまで、変動性の元を除去し続けることです。 2 番目の方法は、測定を可能な限り現実的なものにするよう試み、その結果を統計的に記述することです。 2 番目の方法は、運用の状態と多少相関する結果を生み出すことに注意してください。
- システムは、1 つのプログラムの 1 つのインスタンスを実行するためだけに使用されることはめったにありません。 ほとんど常にデーモンが実行され、頻繁に通信アクティビティーが行われ、複数のユーザーからのワークロードもよくあります。 これらのアクティビティーが直線的に累積されていくことは、まずありません。 例えば、ある特定のプログラムのインスタンスの数を増やしても、そのプログラムの大部分が既にメモリー内にあるために、新しいプログラム・テキストのページはほとんど使用されない結果となります。 しかし、プロセスを追加した結果、プロセッサーのキャッシュのコンテンションが増加するので、その他のプロセスが追加されたプロセスとプロセッサー時間を共用しなければならないだけでなく、すべてのプロセスにおいて命令当たりのサイクル数も増えることになります。 これは実質的に、キャッシュ・ミスがより頻繁に起こる結果として、プロセッサーのスローダウンにつながります。

以下のガイドラインを使用して、特定の状況が許す限り、見積もりを現実的なものにしてください。

- プログラムが存在する場合は、プログラマー自身の要件に最もよく似た既存のインストール・システムで測定します。 最善の方法は、BEST/1 のようなキャパシティー・プランニング・ツールを使用することです。
- 使用できる適切なインストール・システムがない場合は、試験的インストールを行って、疑似ワークロードで測定します。
- 要件にマッチする疑似ワークロードの生成が実際的でない場合は、個々の対話を測定し、その結果をシミュレーションへの入力として使用します。
- プログラムがまだ存在しない場合は、同じ言語および一般的な構造を使用する同等のプログラムを見つけて、それを測定します。 繰り返しますが、言語が抽象的であればあるほど、類似性の判別には注意を要します。
- 同等のプログラムが存在しない場合は、計画した言語で主要なアルゴリズムのプロトタイプを開発し、そのプロトタイプを測定して、ワークロードのモデルを作ります。
- どの種類の測定も実行不可能な場合に限り、経験に基づく推測を行ってください。 計画段階でリソース要件を推測する必要がある場合は、実際のプログラム開発の可能な限り早い段階で、そのプログラムを測定することが重要です。

ISV (independent software vendor) は、アプリケーション評価用のガイドラインを持っていることが多いという点に留意してください。

リソースの見積もりの際、第一の関心事となるのは次の 4 つの局面です (順不同)。

CPU 時間
: ワークロードのプロセッサー・コスト

ディスク・アクセス
: ワークロードがディスクの読み取りまたは書き込みを生成する比率

LAN トラフィック
: ワークロードが生成するパケット数および交換されるデータのバイト数

実メモリー
: ワークロードが必要とする RAM の量

以下の各セクションでは、さまざまな状態で上記の値を決定する方法について説明します。

ワークロード・リソースの測定

実際のプログラム、同等のプログラム、またはプロトタイプを測定に使用できる場合、手法の選択は幾つかの要因に依存します。

以下の要因があります。

- システムが、測定したいワークロードに加えてその他の作業を処理しているかどうか。
- パフォーマンスを低下させる可能性があるツールの使用を許可されているかどうか。例えば、このシステムは実動システムか、または測定の間専用にできるシステムか。
- 本当のワークロードをシミュレートし、監視できる程度。

専用システムでの完全なワークロードの測定:

専用システムの使用が理想的な状態です。システム・オーバーヘッドだけでなく、個々のプロセスのコストも含む測定値を使用できるからです。

システム・アクティビティーの大部分について総合的なシステム・パフォーマンスを測定する際には、以下の **vmstat** コマンドを使用します。

```
# vmstat 5 >vmstat.output
```

これにより、測定の実行中 5 秒ごとにシステムの状態が表示されます。 **vmstat** 出力の最初のセットには、最後のブートから **vmstat** コマンドの開始までの累積データが含まれます。残りのセットは、前の間隔 (このケースでは 5 秒) の結果です。 システム上の **vmstat** 出力は一般に、次のようになります。

```
kthr     memory             page              faults        cpu
----- ----------- ------------------------ ------------ -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 0  1 75186   192   0   0   0   0    1   0 344 1998 403  6  2 92  0
```

CPU およびディスク・アクティビティーを測定する際には、以下の **iostat** コマンドを使用します。

```
# iostat 5 >iostat.output
```

これにより、測定の実行中 5 秒ごとにシステムの状態が表示されます。 **iostat** 出力の最初のセットには、最後のブートから **iostat** コマンドの開始までの累積データが含まれます。残りのセットは、前の間隔 (このケースでは 5 秒) の結果です。 システム上の **iostat** 出力は一般に、次のようになります。

```
tty:      tin         tout   avg-cpu:  % user    % sys     % idle    % iowait
          0.0          0.0              19.4      5.7       70.8       4.1

Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk0           8.0      34.5       8.2         12       164
hdisk1           0.0       0.0       0.0          0         0
cd0              0.0       0.0       0.0          0         0
```

メモリーを測定するには、**svmon** コマンドを使用します。 **svmon -G** コマンドを使用すると、総合的なメモリー使用の状況が表示されます。 統計情報は 4 KB ページ単位です。

```
# svmon -G

               size      inuse       free        pin    virtual
memory        65527      65406        121       5963      74711
pg space     131072      37218

               work       pers       clnt       lpage
pin            5972          0          0           0
in use        54177       9023       2206           0
```

この例では、マシンの 256 MB メモリーが完全に使用されています。 RAM の約 83% は、実行中プログラムの作業セグメントであり、読み取り/書き込みメモリーに使用されます (残りはキャッシング・ファイル用です)。 長時間実行されるプロセスがある場合は、そのメモリー所要量を詳細に検討することができます。 以下の例では、ユーザーのプロセス hoetzel が使用するメモリーを判別します。

```
# ps -fu hoetzel
     UID   PID  PPID   C    STIME    TTY  TIME CMD
 hoetzel 24896 33604   0 09:27:35  pts/3  0:00 /usr/bin/ksh
 hoetzel 32496 25350   6 15:16:34  pts/5  0:00 ps -fu hoetzel

# svmon -P 24896

------------------------------------------------------------------------------
     Pid Command        Inuse      Pin     Pgsp  Virtual   64-bit    Mthrd    LPage
   24896 ksh             7547     4045     1186     7486        N        N        N

  Vsid     Esid Type Description           LPage   Inuse   Pin Pgsp Virtual
     0        0 work kernel seg               -    6324  4041 1186  6324
6a89aa        d work shared library text      -    1064     0    0  1064
72d3cb        2 work process private          -      75     4    0    75
401100        1 pers code,/dev/hd2:6250       -      59     0    -     -
 3d40f        f work shared library data      -      23     0    0    23
16925a        - pers /dev/hd4:447             -       2     0    -     -
```

作業セグメント (5176) は 4 ページを使用しており、**ksh** プログラムのこのインスタンスのコストになります。 共用ライブラリーのコストである 2619 ページはすべての実行中プログラムに分散し、**ksh** プログラムのコスト 58 ページはすべての実行中プログラムと **ksh** プログラムのすべてのインスタンスにそれぞれ分散しています。

256 MB のシステムが必要なサイズより大きいと考える場合は、**rmss** コマンドを使用してマシンの実効サイズを縮小し、ワークロードを測定し直します。 ページングが著しく増加するか、応答時間が低下する場合は、メモリーを減らしすぎています。 この手法は、ワークロードを低下させずに実行できるサイズが検出されるまで、継続することができます。この手法について詳しくは、 154 ページの『rmss コマンドによるメモリー所要量の評価』を参照してください。

ネットワークの使用率を測る基本コマンドは、**netstat** プログラムです。 以下の例は、特定のトークンリング・インターフェースのアクティビティーを示しています。

```
# netstat -I tr0 5
   input    (tr0)     output           input   (Total)    output
 packets  errs  packets  errs colls   packets  errs  packets  errs colls
```

```
35552822 213488 30283693     0     0  35608011 213488 30338882     0     0
     300     0      426     0     0       300     0      426     0     0
     272     2      190     0     0       272     2      190     0     0
     231     0      192     0     0       231     0      192     0     0
     143     0      113     0     0       143     0      113     0     0
     408     1      176     0     0       408     1      176     0     0
```

このレポートの最初の行は、最後のブート以後の累積ネットワーク・トラフィックを示しています。 その後の各行は、前の 5 秒の間隔のアクティビティーを示します。

実動システム上の完全なワークロードの測定:

実動システムでの測定の手法は、専用システムでの手法と同様ですが、システム・パフォーマンスを低下させないように注意する必要があります。

費用対効果の最も高いツールは、**vmstat** コマンドです。このコマンドでは、メモリー、入出力、および CPU 使用率に関するデータが 1 つのレポートで提供されます。 **vmstat** の間隔が妥当な長さ、例えば 10 秒に保たれている場合、平均コストは相対的に低くなります。 **vmstat** コマンドの使用法について詳しくは、 36 ページの『パフォーマンス制約リソースの識別』を参照してください。

実動システム上の部分ワークロードの測定:

部分ワークロードとは、実動システムのワークロードの一部を別のシステムに転送するか、あるいは別のシステムに複製を作成して測定することです。

これは実動システムなので、できる限り控えめに行わなければなりません。 同時に、ワークロードをより詳細に分析して、関心のある部分とそうでない部分を区別する必要があります。部分測定を行うには、関心のあるワークロード・エレメントに共通するものは何かを発見する必要があります。 ワークロード・エレメントについて、以下を確認してください。

- 同じプログラムか、関連したプログラムの小さいセットか。
- システムの 1 ユーザーまたは複数の特定ユーザーによって行われる作業か。
- 1 台以上の特定端末から生じる作業か。

その共通性により、以下のいずれかを使用して、

```
# ps -ef | grep pgmname
# ps -fuusername, . . .
# ps -ftttyname, . . .
```

関心のあるプロセスを識別し、それらのプロセスの累積 CPU 時間使用量を報告することができます。 次に **svmon** コマンドを (賢明に) 使用して、プロセスのメモリー使用量を評価することができます。

個別のプログラム測定:

個々のプログラムのリソース使用量を測定するために、多くのツールを使用することができます。 これらのプログラムの中には、より広範囲のワークロードの測定が可能なものもありますが、実動システムで使用するにはリスクが大きすぎる場合があります。

これらのツールの大部分は、特定リソースの使用量を最小にするためのチューニング方法を説明するセクションで、さらに詳しく説明されています。 よく知られているツールの一部を、以下に示します。

svmon
: プロセスによって使用される実メモリーを測定します。 139 ページの『メモリー使用量』に説明があります。

time 個々のプログラムの経過実行時間および CPU 使用量を測定します。 122 ページの『マイクロプロセッサー使用率測定のための time コマンドの使用法』に説明があります。

tprof プログラム、サブルーチン・ライブラリー、およびオペレーティング・システムのカーネルによる相対 CPU 使用量を測定します。 *Performance Tools Guide and Reference*の『Profiling tools』のセクションに詳しい説明があります。

vmstat -s
プログラムによって生成される入出力ロードを測定します。 203 ページの『vmstat コマンドによる総合的なディスク入出力の評価』に説明があります。

新規プログラムに必要なリソースの見積もり

コーディング・フェーズ中に行われる発明や設計変更は予測を超えるものですが、以下のガイドラインは、要件の概要を理解するのに役立ちます。

まだ作成していないプログラムについて正確な見積もりを行うのは不可能です。出発点として、最小のプログラムには以下のものが必要です。

- 約 50 ミリ秒の CPU 時間、ほとんどがシステム時間
- 実メモリー
  - プログラム・テキスト用に 1 ページ
  - 作業 (データ) セグメント用に 15 ページ (うち 2 ページは固定)
  - `libc.a` へのアクセス。通常これは、他のすべてのプログラムと共用で、オペレーティング・システムの基本コストの一部と見なされます。
- 約 12 のページイン・ディスク入出力操作 (プログラムが最近コンパイル、コピー、または使用されていない場合)。 それ以外の場合は、いずれも必要ありません。

上記に対して、設計に含まれる要求の基本コスト許容量を追加します (与えられている単位は、例として出されているだけです)。

- CPU 時間
  - 高水準の反復または高価なサブルーチン呼び出しを含まない、通常のプログラムの CPU 使用量は、ほとんど計測不能なほどわずかです。
  - 計画中のプログラムが計算上不経済なアルゴリズムを含んでいる場合は、プロトタイプを開発し、そのアルゴリズムを測定してください。
  - 計画中のプログラムが、X または Motif 構成あるいは **printf()** サブルーチンなどの、計算主体の不経済なライブラリー・サブルーチンを使用している場合は、その CPU 使用量を、ほかの部分を小さくしたプログラムによって測定してください。
- 実メモリー
  - プログラム・テキストのページ当たり約 350 行のコードを見積もります。これは、1 行当たり約 12 バイトです。 コーディングのスタイルやコンパイラー・オプションによっては、どちらの方向にも 1 または 2 倍の差が生じうることに留意してください。 この許容量は、代表的なシナリオで扱われるページのためのものです。 まれにしか実行されないサブルーチンを実行可能プログラムの最後に置くような設計では、それらのページは通常実メモリーを消費しません。
  - `libc.a` 以外の共用ライブラリーを参照すると、それらのライブラリーがその他のプログラム、または見積もられているプログラムのインスタンスと共用されていない場合に限って、メモリー所要量が増加します。 これらのライブラリーのサイズを測るには、それらを参照する単純な、長時間実行されるプログラムを作成し、そのプロセスに対して **svmon -P** コマンドを使用します。

- 設計に示されるデータ構造体に必要なストレージの量を見積もります。 最も近いページに切り上げます。
- 短時間の実行では、各ディスク入出力操作は 1 ページのメモリーを使用します。ページは既に使用可能になっているものと想定します。 そのプログラムが、別のプログラムのページが解放されるのを待つとの想定はしないでください。

- ディスク入出力
  - 順次入出力の場合、4096 バイトの読み取りまたは書き込みごとに入出力操作が 1 回行われます。ただし、ファイルがごく最近アクセスされ、そのページの一部がまだメモリー内にある場合は入出力操作を行いません。
  - ランダム入出力の場合、別々の 4096 バイトのページに対するアクセスごとに、それがどれほど小さくても、入出力操作が 1 回行われます。ただし、ファイルがごく最近アクセスされ、そのページの一部がまだメモリー内にある場合は入出力操作を行いません。
  - ラージ・ファイル内の 4 KB ページの各順次読み取りまたは書き込みには、約 100 ユニットを要します。 4 KB ページの各ランダム読み取りまたは書き込みには、約 300 ユニットを要します。実ファイルは、プログラムによって順次に書き込みまたは読み取りが行われるとしても、ディスクに順次に保管されるとは限らないことを覚えておいてください。 したがって、実際のディスク・アクセスの CPU コストは通常、順次アクセスのコストよりランダム・アクセスのコストに近くなります。
- 通信入出力
  - ディスク入出力が実際にネットワーク・ファイルシステム (NFS) のリモート・マウント・ファイルシステムに対して行われる場合、そのディスク入出力はサーバー上で実行されますが、クライアント側でも CPU やメモリーがより多く要求されることになります。
  - どの種類の RPC も、実質的に CPU ロードの一因となります。 設計段階で計画中の RPC は、最小化、バッチ化、プロトタイプ化して、あらかじめ測定しておく必要があります。
  - 4 KB ページの各順次 NFS 読み取りまたは書き込みには、クライアント上で約 600 ユニットを要します。 4 KB ページの各ランダム NFS 読み取りまたは書き込みには、クライアント上で約 1000 ユニットを要します。
  - Web ブラウザーおよび Web サーバーの使用時には、TCP 接続のオープンとクローズが非常に頻繁に行われ、大量のネットワーク入出力が発生することになります。

プログラム・レベルの見積もりからワークロードの見積もりへの変換

ピーク時および通常時のリソース要件を見積もる際の最善の方法は、BEST/1 のようなキューイング・モデルを使用することです。

静的モデルを使用することもできますが、ピーク時のリソースを実際より多く、あるいは少なく見積もるリスクを冒すことになります。 いずれの場合も、ワークロード内の複数プログラムがどのように相互作用するかについて、リソース要件の観点から理解する必要があります。

静的モデルを構築する場合は、最も頻繁に使用される、または要求がきびしいプログラム (通常、この 2 つは同じ) に指定された、許容される最悪の応答時間である時間間隔を使用します。 一般的に、各間隔中にどのプログラムを実行するかを、予定ユーザー数、ユーザーの考慮時間やキー入力の速度、および予想される各種操作の混在を基にして決定してください。

次のガイドラインを使用します。

- CPU 時間
  - 間隔中に実行されるすべてのプログラムの CPU 所要量を合計します。 それらのプログラムが行うディスクおよび通信入出力の CPU 所要量を含めます。

- この数値が、間隔中に使用可能な CPU 時間の 75% より大きい場合は、ユーザーを減らすか、CPU を増やすことを考慮してください。

- 実メモリー
  - オペレーティング・システムのメモリー所要量は、物理メモリーの量に比例します。 オペレーティング・システム自体のための 6 MB から 8 MB から始めてください。 小さい方の数字は、スタンドアロン・システム用です。 後の数字は、LAN 接続され、TCP/IP および NFS を使用するシステム用です。
  - 間隔中に実行されるプログラムのすべてのインスタンスの作業セグメント所要量を、プログラムのデータ構造体用のスペース見積もりを含めて合計します。
  - その合計に、実行される別個の各プログラムのテキスト・セグメントのメモリー所要量を加算します (プログラム・テキストの 1 コピーは、そのプログラムのすべてのインスタンスの要求を満たします)。 非共用ライブラリーからのどのサブルーチンも (そしてそのサブルーチンのみが) 実行可能プログラムの一部となりますが、非共用ライブラリー自体はメモリー内に入れられないことを覚えておいてください。
  - その合計に、ワークロード内のプログラムによって使用される共用ライブラリーのそれぞれが消費するスペースの量を加算します。 繰り返しますが、1 つのコピーがすべての要求を満たします。
  - 何らかのファイル・キャッシングおよびフリー・リストに十分なスペースを使用できるように、合計メモリーの計画は、使用するマシンのサイズの 80% を超えないようにしてください。

- ディスク入出力
  - 各プログラムの各インスタンスから発生する入出力の数を追加します。 入出力は、小さいファイルへの入出力 (またはラージ・ファイルへのランダム・アクセス) 合計数、および純粋にラージ・ファイル (32 KB を超えるもの) への順次読み取りまたは書き込みの合計数に分けます。
  - 十分と考えられる入出力をメモリーから減じます。 直前の間隔で読み取りまたは書き込みが行われたどのレコードも、多くの場合、現在の間隔でまだ使用可能です。 そのほかに、検討中のマシンのサイズとマシンのワークロードの RAM 所要量の合計を調べます。 オペレーティング・システムの所要量とワークロードの所要量を引いた残りのスペースには、多くの場合、最新の読み取りまたは書き込みファイル・ページが含まれます。 アプリケーションの設計で、最近アクセスしたデータを再利用する可能性が非常に高い場合は、キャッシングの効果を考慮に入れて計算することができます。再利用はページ・レベルであって、レコード・レベルではないことを忘れないようにしてください。特定のレコードの再利用の可能性が低い場合で、ページ当たりのレコード数が多い場合は、所定の間隔に必要なレコードの一部が、最近使用された他のレコードと同じページに入ると考えられます。
  - 正味の入出力所要量 (ディスクごとの 1 秒当たりのディスク入出力) を現行ディスク・ドライブの近似の能力と比較します。 ランダムまたは順次所要量がアプリケーション・データを保持するディスクの、対応する能力の合計の 75% より大きい場合は、アプリケーションが実動状態になったとき、チューニング (あるいは拡張も) が必要になります。

- 通信入出力
  - ワークロードの帯域幅使用量を計算します。 LAN 上のすべてのノードの帯域幅使用量の合計が、名目帯域幅の 70% (イーサネットの場合は 50%) より大きい場合は、より高い帯域幅のネットワークを使用する必要があります。
  - サーバー上に置かれる追加ロードの CPU、メモリー、および入出力所要量について同様の分析を行います。

注: 上記のガイドラインは、徹底的な測定が不可能な場合にのみ 使用するためのものであることを忘れないようにしてください。 ガイドラインの代わりにアプリケーション固有の測定を使用する方が、見積もりの正確度が相当に向上します。

効率的なプログラム設計とインプリメンテーション

どのリソースがプログラムの速度を制限するかの判別が終了したら、そのリソースの使用を最小化するための適切な手法について説明するセクションに直接進むことができます。

それ以外の場合は、プログラムのバランスがとれること、およびこのセクションでの推奨事項をすべて適用することを前提としてください。 プログラムがインプリメントされたら、 36 ページの『パフォーマンス制約リソースの識別』に進んでください。

プロセッサー制約プログラム

プログラムのほとんどすべてが数値計算からなっているために、このプログラムがプロセッサー制約の場合は、選択したアルゴリズムがプログラムのパフォーマンスに大きな影響を与えます。

本当のプロセッサー制約プログラムの最大速度は、以下のものによって決まります。

- 使用するアルゴリズム
- プログラマーが作成するソース・コードおよびデータ構造体
- コンパイラーが生成するマシン言語命令のシーケンス
- プロセッサーのキャッシュのサイズおよび構造
- プロセッサー自体のアーキテクチャーおよびクロック速度 (452 ページの『マイクロプロセッサー速度の判別』を参照)

代替アルゴリズムについての説明は、この資料で取り扱う範囲を超えています。 本書は、計算効率についてはアルゴリズムの選択の際に考慮されているという前提で書かれています。

アルゴリズムが決まると、前述のリストでプログラマーが影響を与えることのできる項目は、ソース・コード、使用するコンパイラー・オプション、および場合によってはデータ構造体だけです。 以下の各セクションでは、ユーザーがソース・コードを持っている個々のプログラムの効率を改善するために使用できる手法を取り扱います。 ソース・コードが利用不能の場合は、チューニングまたはワークロード・マネージメント手法の使用を試みてください。

キャッシュの有効使用のための設計およびコーディング

ストレージの有効使用とは、使用される見込みの命令およびデータでストレージを満たしておくことです。

プロセッサーには、以下のように多層のメモリー階層があります。

1. 命令パイプラインおよび CPU レジスター
2. 命令キャッシュとデータ・キャッシュ、および対応する変換索引バッファー
3. RAM
4. ディスク

命令およびデータは、この階層を上がるにつれて、その下のレベルより高速のストレージに移動しますが、サイズは小さくなり、費用もより多くかかります。 したがって、所定のマシンから考えられる最高のパフォーマンスを得るには、プログラマーは、各レベルで選択可能なストレージを最も効率よく使用する必要があります。

効率的なストレージ使用の障害となるのは、通常は、ストレージがプログラムまたはデータ構造体内の境界に対応しない、キャッシュ・ラインや実メモリー・ページなどの固定長ブロックに割り当てられているという事実です。 ストレージ階層とは無関係に設計されるプログラムおよびデータ構造体では、負荷の小さいシステムまたは負荷の重いシステムでパフォーマンスの効果が逆に働き、割り当てられたストレージの使用が非効率的になることがよくあります。

ストレージ階層を考慮に入れるということは、キャッシュまたは仮想メモリー環境における効率的なプログラミングの一般原則を理解し、それに適合させることです。 再パッケージ化手法は、再コーディングしなくても有効な改善をもたらすので、新しいコードを設計するときは、ストレージの効率的な使用に留意して行う必要があります。

階層ストレージの効率的な使用についてのディスカッションでは、参照の局所性 および作業セット の 2 つの用語が不可欠です。

- プログラムの参照の局所性とは、その命令実行アドレスやデータへの参照が、一定の時間間隔内に小さいストレージ域にクラスター化される度合いのことです。
- その同じ間隔内のプログラムの作業セットとは、使用中の一組の記憶ブロック、あるいはより正確にいえば、それらのブロックを占めるコードまたはデータのことです。

参照の局所性に優れたプログラムは、最小の作業セットしか持ちません。これは、使用中のブロックには実行コードまたはデータが密にまとまっているからです。 参照の局所性に乏しい、機能的に同等のプログラムは、より大きい作業セットを持っています。これは、より広い範囲のアドレスがアクセスされるのに合わせて、より多くのブロックが必要になるからです。

各ブロックを所定のレベルの階層にロードするにはかなりの時間がかかるので、階層ストレージ・システムの効率的なプログラミングの目標は、作業セットを現実的に可能な範囲で小さく保てるように、コードを設計しパッケージすることです。

次の図は、サブルーチン・レベルでのよい例と悪い例を示しています。 プログラムの最初のバージョンは、多くの場合プログラムを書いた順序でパッケージされています。 最初のサブルーチン **PriSub1** には、プログラムのエントリー・ポイントが含まれます。 このサブルーチンは常に 1 次サブルーチン **PriSub2** および **PriSub3** を使用します。このプログラムのあまり使用されない機能の中に、2 次サブルーチン **SecSub1** および **SecSub2** を必要とするものがあります。 まれな場合ですが、エラー・サブルーチン **ErrSub1** および **ErrSub2** が必要になることもあります。

参照の局所性がよくない、大きな実効ページ・セット

ページ 1			ページ 2			ページ 3
PriSub1	SecSub1	ErrSub1	PriSub2	SecSub2	ErrSub2	PriSub3

参照の局所性がよい、小さな実効ページ・セット

ページ 1			ページ 2			ページ 3
PriSub1	PriSub2	PriSub3	SecSub1	SecSub2	ErrSub1	ErrSub2

図 *15.* 参照の局所性： 図の上半分は、参照の局所性が少ないバイナリー・プログラムがどのようにパッケージされているかを示しています。 バイナリーの実行可能プログラム内では、最初に **PriSub1** の命令があり、その後 **SecSub1**、**ErrSub1**、**PriSub2**、**SecSub2**、**ErrSub2**、および **PriSub3** の命令が続きます。 この実行可能ファイルのうち、**PriSub1**、**SecSub1**、および **ErrSub1** の命令はメモリーの 1 ページ目に入っています。 **PriSub2**、**SecSub2**、**ErrSub2** の命令はメモリーの 2 ページ目、**PriSub3** の命令はメモリーの 3 ページ目に入っています。 **SecSub1** と **SecSub2** の使用頻度は低く、また **ErrSub1** と **ErrSub2** もほとんど使用されません。したがって、このプログラムのパッケージ方法は参照の局所性が少なく、必要以上のメモリーを使用する可能性があります。 図の下半分では、**PriSub1**、**PriSub2**、および **PriSub3** が隣り合わせに配置され、メモリーの 1 ページ目に入っています。 **PriSub3** の後には **SecSub1**、**SecSub2**、および **ErrSub1** が続き、これらはすべてメモリーの 2 ページ目に入っています。 最後に、**ErrSub2** が末尾に配置され、メモリーの 3 ページ目に入っています。 **ErrSub2** が必要になることはまずないので、このケースではメモリー所要量が 1 ページ減ることになります。

プログラムの初期バージョンでは、通常の場合で実行に 3 ページのメモリーを要するので、参照の局所性に優れているとは言えません。 2 次サブルーチンおよびエラー・サブルーチンによって、プログラムのメインパスが 3 つの、物理的に離れたセクションに分離されます。

プログラムの改良バージョンでは、1 次サブルーチンを相互に隣接して配置し、使用頻度の低い機能をその次に置いています。 必要なエラー・サブルーチン (ほとんど使用されない) は、実行可能プログラムの終わりに配置します。 これで、以前は 3 ページのメモリーが必要だった、プログラムの最もよく使う機能を、1 回のディスク読み取りおよび 1 ページのメモリーで取り扱うことができるようになりました。

参照の局所性および作業セットは、時間を基準にして定義されることを覚えておいてください。 プログラムが段階的に動作する場合は、ステージごとに相当の時間がかかり、しかもサブルーチンの別々のセットを使用する場合は、各ステージの作業セットの最小化を試みてください。

レジスターおよびパイプライン

一般に、レジスター・スペースの割り当てと最適化を行うこと、パイプラインをフルに保持することはコンパイラーの役割です。

プログラマーの主な義務は、コンパイラー最適化手法を無効にするような構造を回避することです。 例えば、プログラムの重要なループの 1 つにサブルーチンの 1 つを使用する場合、実行時間を最小化するためには、コンパイラーがそのサブルーチンをインラインにすることが適切と思われます。 しかし、サブルーチンが別の C モジュールにパッケージされている場合、それをコンパイラーがインラインにすることはできません。

キャッシュおよび TLB

キャッシュには Translation Lookaside Buffers (TLB) があります。これには、命令テキストまたはデータの最近使用されたページの仮想アドレスから実アドレスへのマッピングが含まれます。

プロセッサーのアーキテクチャーおよびモデルによって異なりますが、プロセッサーは、以下のものを保持するための 1 個から数個のキャッシュを持っています。

- 実行プログラムの一部
- 実行プログラムが使用するデータ
- TLBs

キャッシュ・ミスが発生すると、完全なキャッシュ線をロードするのに数十のプロセッサー・サイクルを要することがあります。 TLB ミスが発生すると、仮想ページから実ページへのマッピングを計算するのに数十のサイクルを要することがあります。 正確なコストは、インプリメンテーションに依存します。

プログラムとデータがキャッシュに収まるとしても、線または TLB エントリーを多数使用するほど (つまり、参照の局所性が低いほど)、すべてをロードするのにより多くの CPU サイクルを要します。 命令やデータを何度も再利用しなければ、ロードする際のオーバーヘッドがプログラム実行時間の合計のかなりの部分を占め、結果としてシステム・パフォーマンスが低下します。

優れたプログラミング手法では、プログラムのメインライン、すなわち標準的なフローを可能な限りコンパクトに保ちます。 メインのプロシージャーおよびそれが頻繁に呼び出すすべてのサブルーチンは、隣接している必要があります。 不明瞭なエラーなど、確率の低い条件は、メインラインでのみテスト対象とすべきです。 条件が実際に起こった場合、その処理は別個のサブルーチンで行う必要があります。 そのようなサブルーチンはすべて、モジュールの終わりにひとまとめにしてください。 こうした配置により、使用率の低いコードが使用率の高いキャッシュ線内のスペースを占有する確率が下がります。 大きなモジュールでは、使用率の低いサブルーチンの一部または全部が、ほとんどメモリーに読み込む必要のないページを占める可能性があります。

同じ原則がデータ構造体にも適用されます。ただし、データ・レイアウトに関するコンパイラーの規則を補正するために、時にはコードの変更が必要となります。

例えば、行列乗算のような一部の行列演算は、極度に単純化してコーディングした場合、参照の局所性に乏しいアルゴリズムになります。 行列演算では一般に、行エレメントが列エレメントに作用するというように、行列データに順次にアクセスします。各コンパイラーは、行列のストレージ・レイアウトについて特定の規則を持っています。 FORTRAN コンパイラーでは、行列を、列を主としたフォーマットでレイアウトします (つまり、列 1 のすべてのエレメント、次に列 2 のすべてのエレメントというように)。 C コンパイラーでは、行列を、行を主としたフォーマットでレイアウトします。 行列が小さい場合、行エレメントと列エレメントはデータ・キャッシュに収めることができ、プロセッサーおよび浮動小数点ユニットはフルスピードで稼働することができます。 ただし、行列のサイズが大きくなるにつれて、そうした行/列演算の参照の局所性は、データをキャッシュ内に維持できなくなる程度まで低下します。実際、行/列演算の本来のアクセス・パターンでは、アクセスされる一連のエレメントがキャッシュより大きい場合に、キャッシュのスラッシング・パターンを生成し、最初にアクセスされたエレメントをキャッシュから出した後、再度同じデータについてアクセス・パターンを繰り返します。

この種の行列アクセス・パターンの一般的なソリューションは、演算をブロックに分割し、エレメントがキャッシュ内にとどまっている間に、同じエレメント上で複数の演算を実行できるようにすることです。 この一般的な手法には、ストリップ・マイニング *(strip mining)* という名前が付けられています。

数値解析のエキスパートが、ストリップ・マイニングおよびその他の最適化手法を使用する、行列操作アルゴリズムの複数のバージョンをコーディングするよう依頼されました。 その結果、行列操作のパフォーマンスが 30 倍向上しました。 チューニング済みのルーチンは、Basic Linear Algebra Subroutine (BLAS) ライブラリー `/usr/lib/libblas.a` にあります。パフォーマンス・チューニング済みサブルーチンの大きい方のセットは、Engineering and Scientific Subroutine Library (ESSL) ライセンス・プログラムです。

基本線形代数サブルーチンの機能およびインターフェースは、「*AIX Version 7.1 Technical Reference*」に記載されています。このライブラリーを使用するには、FORTRAN ランタイム環境がインストールされていなければなりません。ユーザーは一般に、その行列およびベクトル演算にこのライブラリーを使用すべきです。それは、そのサブルーチンが、ユーザーが自分では達成できそうにない程度にまでチューニングされているからです。

データ構造体がプログラマーによって制御されている場合は、その他の効率化が可能です。 一般原則は、可能な場合は必ず、頻繁に使用されるデータをひとまとめにするというものです。 構造体に、頻繁にアクセスされる制御情報および時々アクセスされる詳細データが含まれる場合は、制御情報が連続バイトに割り当てられていることを確認してください。 これにより、すべての制御情報が、単一の (または少なくとも最小数の) キャッシュのミスで、キャッシュにロードされる確率が高まります。

プリプロセッサーおよびコンパイラーの使用状況

命令の再配列において、コンパイラーに異なる自由度を与える、幾つかの最適化レベルがある。

所定のマシンで実行される所定のプログラムから、可能な最高のパフォーマンスを得たいプログラマーは、次のような幾つかの考慮事項を検討する必要があります。

- 一部のソース・コード構造を再配置して、より効率的な実行可能コードにコンパイルできる、機能的に同等のソース・モジュールを形成することが可能なプリプロセッサーがある。
- アーキテクチャーに幾つかの変種があるように、特定の変種または変種の集合の最適コンパイルを可能にする、幾つかのコンパイラー・オプションがある。
- プログラマーは、**#pragma** フィーチャーを使用して、最悪のケースの前提事項の一部を緩和することにより、コンパイラーがより効率的なコードを生成できるようにする、プログラムのある特定の局面を C コンパイラーに通知することができる。

実験することのできないプログラマーは、常に最適化を行う必要があります。 最適化されたコードと最適化されていないコードのパフォーマンスの差はほとんど常に非常に大きいので、常に基本最適化 (コンパイラー・コマンドの **-O** オプション) を使用するようにしてください。 例外は、単純明快なコード生成を特に必要とする、テスト状態の場合だけです (例えば、**tprof** ツールを使用したステートメント・レベルのパフォーマンス分析など)。

これらの手法は、プログラムによってはさらにパフォーマンスの向上をもたらしますが、特定プログラムについて、どのような組み合わせが最高のパフォーマンスをもたらすかを判別するには、相当の再コンパイルおよび測定を必要とする可能性があります。

コンパイラーを効率的に使用するための手法について、詳しくは「*Optimization and Tuning Guide for XL Fortran, XL C and XL C++*」を参照してください。

最適化レベル

コンパイラーが生成するコードを最適化する度合いは **-O** フラグで制御されます。

最適化なし
: **-O** フラグのどのバージョンも存在しない場合、コンパイラーは、パフォーマンスの改善時に命令の再配列もその他の試みも行わずに単純明快なコードを生成します。

-O または **-O2**

これらのフラグを使用した場合、コンパイラーは、コードの再配列に関する控えめな前提事項に基づいて最適化を行います。 **#pragma** ディレクティブのような明示的緩和のみが使用されます。このレベルでは、ソフトウェアのパイプライン処理、ループのアンロール、または単純な予測による共通副次式の除去は行われません。 また、コンパイラーが使用できるメモリーの量にも制約があります。

-O3 このフラグは、コンパイラーに対して、使用する最適化手法に関して積極的になるように、また最大限の最適化を行うのに必要なだけのメモリーを使用するように指示します。プログラムが浮動小数点例外、ゼロの符号、または再配列計算の精度の結果に敏感な場合は、このレベルの最適化の結果として、プログラムに機能上の変更が必要になることがあります。 多少パフォーマンスが犠牲になりますが、上記の副次作用は、**-qstrict** オプションを **-O3** と組み合わせて使用することにより回避することができます。 **-qhot** オプションを **-O3** と組み合わせて使用すると、予測による共通副次式の除去と多少のアンロールが使用可能になります。 上記の変更の結果、大きなルーチンまたは複雑なルーチンのパフォーマンスは、コンパイラーが従来のバージョンで **-O** オプションとともに持っていた **-O3** オプションを (あるいは **-qstrict** または **-qhot** と組み合わせて) 使用した場合と同じかそれ以上に改善されるはずです。

-O4 このフラグは **-O3 -qipa** と同等で、そのプラットフォームに最適なアーキテクチャーとチューニング・オプションが自動生成されます。

-O5 このフラグは **-O4** と類似しています。ただし、この場合、**-qipa = level = 2** は例外です。

ハードウェア・プラットフォームのコンパイル

特定のハードウェア・プラットフォーム用にコンパイルするには、多くの考慮すべき項目があります。

システムは幾つかのタイプのプロセッサーを使用することができます。 **-qarch** および **-qtune** オプションを使用することによって、これらのプロセッサーの特殊な命令および特定の強度に関してプログラムを最適化できます。

以下のガイドラインに従ってください。

- プログラムを単一のシステムでのみ、または同じプロセッサー・タイプのシステムのグループでのみ実行する場合には、**-qarch** オプションを使用して、プロセッサー・タイプを指定してください。
- プログラムを複数の異なるプロセッサー・タイプを持つシステムで実行し、かつ、最も重要なプロセッサー・タイプを 1 つ識別できる場合には、該当する **-qarch** と **-qtune** の設定値を使用してください。FORTRAN および HPF ユーザーは、**xxlf** および **xxlhpf** コマンドを使用して、これらの設定を対話式に選択することができます。
- プログラムをすべての範囲のプロセッサー・インプリメンテーションで実行する予定であり、特定のプロセッサー・タイプを主に使用することがない場合には、**-qarch** も **-qtune** も使用しないでください。

string.h サブルーチンのパフォーマンスの C オプション

オペレーティング・システムは、string サブルーチンを `libc.a` から使用するのではなく、アプリケーション・プログラムに組み込む能力を提供して、呼び出し時間とリターン・リンケージ時間を節約します。

string サブルーチンを組み込むには、サブルーチンの使用に先立って、アプリケーションのソース・コードに以下のステートメントを入れておく必要があります。

```
#include <string.h>
```

最高のパフォーマンスを得るための C および C++ コーディング・スタイル

多くの場合、C 構成のパフォーマンス・コストは明らかでなく、ときには直観に反したものでさえあります。

これらの状態の幾つかを、以下に示します。

- 可能な場合は必ず、*char* または *short* の代わりに *int* を使用してください。

  ほとんどの場合、*char* および *short* データ項目を使用するには、より多くの命令を操作する必要があります。 余分な命令には時間がかかる上、大きな配列の場合を除いて、より小さいデータ型を使用することによってスペースを節約するほうが、実行可能プログラムのサイズを増加して補正することより優っています。

- *char* を使用する必要がある場合は、可能であれば、それを *unsigned* にしてください。

  signed char の場合は、この変数がレジスターにロードされるつど、*unsigned char* より 2 つ余分に命令が必要になります。

- 可能な場合は必ず、グローバル変数ではなく、ローカル (自動) 変数を使用してください。

  グローバル変数は、アクセスするのにローカル変数より多くの命令を必要とします。 また、逆に情報が存在しない場合、コンパイラーは、グローバル変数がサブルーチン呼び出しによって変更されていると想定します。サブルーチン呼び出しの後でグローバル変数の値を使用する場合は、その値を再ロードする必要があるので、この変更は最適化には逆効果です。

- グローバル変数 (他のスレッドと共用されていない) にアクセスする必要がある場合は、その値をローカル変数にコピーして、そのコピーを使用してください。

  グローバル変数へのアクセスが 1 回のみでない限り、ローカル・コピーを使用した方が効率的です。

- 状態を記録し、テストする際は、文字列ではなくバイナリー・コードを使用してください。文字列は、データと命令スペースの両方を消費します。 例えば、次のシーケンス、

```
#define situation_1 1
#define situation_2 2
#define situation_3 3
int situation_val;

situation_val = situation_2;
. . .
if (situation_val == situation_1)
. . .
```

  は、次のシーケンスよりはるかに効率的です。

```
char situation_val[20];

strcpy(situation_val,"situation_2");
. . .
if ((strcmp(situation_val,"situation_1"))==0)
. . .
```

- 文字列が必要なときは、可能な場合は必ず、null 終了の可変長文字列ではなく、固定長文字列を使用してください。

  memcpy() など、ルーチンの **mem*()** ファミリーは、**strcpy()** などの対応する **str*()** ルーチンより高速です。それは、**str*()** ルーチンは null があるかどうか各バイトを調べる必要があり、**mem*()** ルーチンにはその必要がないからです。

コンパイラーの実行時間

コンパイラーの実行時間に影響を与える幾つかの要因があります。

オペレーティング・システムで、C コンパイラーは 2 つの異なるコマンド、**cc** および **xlc** によって起動することができます。 伝統的にシステムの C コンパイラーを起動するために使用されてきた **cc** コマンドを用いると、C コンパイラーは **langlevel=extended** モードで稼働します。 このモードを使用すると、ANSI 準拠でない既存の C プログラムをコンパイルすることができます。 これもプロセッサー時間を消費します。

コンパイルするプログラムが ANSI 準拠の場合は、**xlc** コマンドを使用して C コンパイラーを起動した方が効率的です。

-O3 フラグを使用すると、暗黙的に **-qmaxmem** オプションが含まれます。このオプションを使用すると、コンパイラーは、最大の最適化を行うのに必要なだけのメモリーを使用することができます。 この状態には、次の 2 つの影響があります。

- マルチユーザー・システムにおいて、大規模な **-O3** コンパイルを行うと、その他のユーザーが経験するパフォーマンスに逆効果になるほど多量のメモリーを消費する場合がある。
- 実メモリーが小さいシステムにおいて、大規模な **-O3** コンパイルを行うと、ページング率が高くなる原因となるほど多くのメモリーが消費され、コンパイルが遅くなることがある。

メモリー制約プログラム

プログラマーは、アドレッシングの制限と苦闘することに慣れていました。例えば、DOS 環境では、256 MB の仮想メモリー・セグメントは実際上無限に思われます。 プログラマーは、ストレージの制約や最小のパス長さと最大限の単純さでコーディングすることを無視したい誘惑にかられます。 あいにく、こうした姿勢には難点があります。

仮想メモリーは大きくても、その速度は可変です。 メモリーを多く使用すればそれだけ速度は遅くなり、しかもその関係は直線的ではありません。 実際にすべてのプログラムによってタッチされる仮想記憶域の合計量 (つまり、作業セットの合計) がマシン内の固定されていない実メモリーの量よりわずかに少ない間は、仮想メモリーは実メモリーとほぼ同じ速度で作動します。 すべての実行プログラムの作業セットの合計が、使用可能なページ・フレームの数を超えると、メモリーのパフォーマンスは最大 2 桁ずつ、急速に低下します (VMM メモリーのロード制御がオフにされている場合)。 システムがこの段階に達した場合、そのシステムはスラッシング しているといわれます。 システムはそのほとんどすべての時間をページングに費やし、各プロセスがその作業セットを入れるために必要なストレージを他のプロセスから取り戻そうとするので、有効な処理は行われません。 VMM メモリー・ロード制御がアクティブの場合、VMM はこの無限に継続しうるスラッシングを回避することができますが、応答時間が著しく増加するという犠牲を払うことになります。

メモリーの非効率的な使用が原因の性能の低下は、キャッシュの非効率的な使用による場合よりはるかに大きくなります。これは、メモリーとディスクの間の速度の差がキャッシュとメモリーの間の差よりずっと大きいからです。キャッシュ・ミスが数十 CPU サイクルを要するのに対して、ページ・フォールトは一般に 10 ミリ秒以上かかりますが、これは少なくとも 400 000 CPU サイクルになります。

VMM メモリー・ロード制御によって、初期のスラッシング状態が無限に継続されないようにすることはできますが、不要なページ・フォールトによる応答時間の低下とスループットの減少という犠牲は避けられません (160 ページの『schedo コマンドによる VMM メモリー・ロード制御チューニング』を参照してください)。

ページング可能コードの構造化:

プログラムのコード作業セットを最小化する際、一般的な目標は、頻繁に実行されるコードを小さい領域にまとめて、めったに実行されないコードと分離することです。

特に、次の点に注意してください。

- エラー処理コードの長いブロックを連続して入れない。 それらを別個のサブルーチンに、できれば別個のソース・コード・モジュールに入れてください。 これは、エラー・パスだけでなく、めったに使用されないすべての機能オプションに適用されます。
- ロード・モジュールを予期せずに構造化しない。 頻繁に呼び出されるオブジェクト・モジュールを、できるだけその呼び出し元に近いところに配置するようにしてください。 めったに呼び出されないサブルーチンから構成されるオブジェクト・モジュールは、 (理想的には) ロード・モジュールの終わりに集中させてください。 それらのオブジェクト・モジュールが位置するページは、まず読み取られることはありません。

ページング可能データの構造化:

データ作業セットを最小化するには、頻繁に使用されるデータを集中させ、仮想記憶域のページに対する不要な参照を行わないようにしてください。

特に、次の点に注意してください。

- **malloc()** または **calloc()** サブルーチンを使用して、実際に必要な量のスペースだけを要求します。 実際にはその一部しか使用しないのに、最大サイズの配列を要求して初期化することは決してしないでください。 その配列エレメントを初期化するために新しいページにタッチすると、事実上、VMM にほかのプロセスから実メモリーのページをスチールするよう強制することになります。 その結果、後でそのページを所有していたプロセスが再度そのページにアクセスしようとしたときに、ページ・フォールトが起こります。 **malloc()** サブルーチンと **calloc()** サブルーチンの違いは、インターフェースだけにあるのではありません。
- **calloc()** サブルーチンは割り当てられたストレージをゼロにリセットするので、割り当てられているすべてのページにタッチするのに対して、**malloc()** サブルーチンは最初のページだけにタッチします。 **calloc()** サブルーチンを使用して大きい領域を割り当て、その先頭の小さい部分だけを使用する場合は、システムに不要な負荷をかけることになります。 それらのページを初期化する必要があるだけでなく、その実メモリー・フレームが再利用される場合は、初期化されただけで決して使用されることのないページをページング・スペースに書き出さなければなりません。 この状態では、入出力スロットとページング・スペース・スロットの両方がむだになります。
- 大型の構造体 (バッファーなど) のリンク・リストも、同様の問題につながることがあります。 プログラムが特定のキーを探すために一連の検索を多く行う場合は、リンクとキーをデータとは別に維持するか、代わりにハッシュ・テーブル・アプローチを使用してください。
- 参照の局所性は、アドレス・スペース内だけでなく、時間の局所性も意味します。データ構造体を使用するときは (使用することがあったとして)、その直前にデータ構造体を初期化してください。 負荷の大きいシステムでは、初期化と使用の間に長時間常駐しているデータ構造体には、そのフレームをスチールされるリスクがあります。 その後プログラムがそのデータ構造体の使用を開始したときに、不要なページ・フォールトが発生することになります。
- 同様に、プログラムの初めの方では使用されただけで、プログラムの後の方ではタッチされないままの大型の構造体は解放する必要があります。 **malloc()** または **calloc()** サブルーチンによって割り当てられたスペースを解放するには、**free()** サブルーチンを使用するだけでは不十分です。 **free()** サブルーチ

ンは、構造体が占めていたアドレス範囲を解放するだけです。 実メモリーとページング・スペースを解放する場合は、**disclaim()** サブルーチンを使用して、スペースも同様に放棄してください。 **disclaim()** は、**free()** より先に呼び出す必要があります。

固定ストレージの誤用**:**

循環性およびタイムアウトを回避するためには、システムの一部を実メモリーに固定する必要があります。

このコードおよびデータについては、作業セットの概念は無意味です。それは、固定された情報はすべて、使用されるかどうかにかかわらず、常に実記憶域内にあるからです。 コードまたはデータを固定するプログラム (ユーザー作成デバイス・ドライバーなど) はどれも、固定ストレージの最低限の量だけが使用されるように、注意深く設計する (移植する場合は綿密に調査する) 必要があります。 以下に、注意を要する例を幾つか示します。

- コードはロード・モジュール (実行可能ファイル) 単位で固定されます。 コンポーネントに固定する必要があるオブジェクト・モジュールとページング可能なオブジェクト・モジュールがある場合は、固定するオブジェクト・モジュールを別のロード・モジュールにパッケージしてください。
- 単に問題があるかもしれないという理由で、モジュールまたはデータ構造体を固定するのは、無責任です。 設計者は、情報が必要とされる条件を理解し、その時点でページ・フォールトを許容するかどうか推察する必要があります。
- 必要なサイズがロードに依存している固定された構造体 (バッファー・プールなど) は、システム管理者によるチューニングが可能でなければなりません。

パフォーマンス関連のインストール・ガイドライン

インストール・プロセスの事前および実行の際について、考慮すべき多くの問題があります。

オペレーティング・システムのプリインストール・ガイドライン

次の 2 つの状態を考慮する必要があります。

- 新規システムへのオペレーティング・システムのインストール

  インストール・プロセスを開始する前に、ディスク・ファイルシステムおよびページング・スペースのサイズとロケーションについて決定していること、およびそれらの決定をオペレーティング・システムに伝える方法を理解していることを確認してください。
- 新しいレベルのオペレーティング・システムの既存のシステムへのインストール

  新しいレベルのオペレーティング・システムにアップグレードする場合は、以下を行ってください。
  - `/etc/tunables/nextboot` ファイルを使用しているかどうかを確認します。
  - やはり `/etc/tunables/nextboot` ファイルを使用する場合は、最初のリブート後に `/etc/tunables/lastboot.log` ファイルを検査します。

マイクロプロセッサーのプリインストール・ガイドライン

マイクロプロセッサーのスケジューリングには、タイム・スライス期間などのデフォルトのパラメーターを使用します。

ほぼ同一の構成で、同じワークロードを使用してモニターやチューニングを行った経験が豊富にあるのでなければ、これらのパラメーターはインストール時には未変更のままにしておいてください。

インストール後の推奨事項については、 112 ページの『マイクロプロセッサーのパフォーマンス』を参照してください。

メモリーのプリインストールのガイドライン

実際のワークロードに対するシステムの応答について経験を積むまで、メモリーしきい値の変更は行わないでください。

インストール後の推奨事項については、 138 ページの『メモリー・パフォーマンス』を参照してください。

ディスクのプリインストールのガイドライン

論理ボリュームを定義し拡張するためのメカニズムは、可能な最善のデフォルト選択を行うようになっています。 しかし、満足のいくディスク入出力パフォーマンスが得られる可能性は、システムのインストーラーが論理ボリュームのサイズと配置を、予期されるデータ・ストレージおよびワークロード要件に合わせて調製した場合の方がはるかに高くなります。

以下の手順に従うことをお勧めします。

- 可能なら、デフォルトのボリューム・グループの rootvg は、システムが最初にインストールされる物理ボリュームのみからなるようにします。 システム内のその他の物理ボリュームを制御するために、1 つ以上の他のボリューム・グループを定義します。この推奨事項には、システム管理に加えて、パフォーマンス、利点も含まれます。
- ボリューム・グループが複数の物理ボリュームで構成される場合は、以下を行うとパフォーマンスが向上することがあります。
  - 最初は、ボリューム・グループを単一の物理ボリュームで定義する。
  - その新規ボリューム・グループ内に論理ボリュームを定義する。 この定義により、最初の物理ボリュームに、ボリューム・グループのジャーナル論理ボリュームが割り当てられます。
  - 残りの物理ボリュームをボリューム・グループに追加する。
  - 新たに追加された物理ボリュームに、高アクティビティーのファイルシステムを定義する。
  - ジャーナル論理ボリュームを含む物理ボリュームに、きわめて低アクティビティーのファイルシステムのみを (ある場合) 定義する。 これは、入出力によりジャーナル・ファイルシステム (JFS) のログ・トランザクションが生じた場合にのみ、パフォーマンスに影響を与えます。

    この方法では、ジャーナル入出力アクティビティーが高アクティビティーのデータ入出力から分離され、オーバーラップの確率が増大します。 この手法は、NFS サーバーのパフォーマンスに特に重大な影響を与える場合があります。データおよびジャーナルの両方の書き込みが、NFS が書き込み操作のための入出力の完了を知らせる前に完了していなければならないからです。
- 最も早い機会に、論理ボリュームをその予期される最大サイズに定義または拡張します。 パフォーマンスが重要な論理ボリュームが、隣接して望ましいロケーションに置かれる確率を最大化するには、それらを最初に定義し、拡張することです。
- 使用頻度の高い論理ボリュームは、複数のディスク・ドライブに分散させる必要があります。 SMIT プログラムの「Add a Logical Volume (論理ボリュームの追加)」画面の「**RANGE of physical volumes** (物理ボリュームの範囲)」オプション (高速パス: **smitty mklv**) が **maximum** に設定されると、新規論理ボリュームはボリューム・グループ (または明示的にリストされた物理ボリュームのセット) の物理ボリューム間で分割されます。
- システムにタイプの異なるドライブがある場合 (またはどのドライブをオーダーするか決めようとしている場合) は、以下のガイドラインを考慮してください。
  - 通常は順次にアクセスされるラージ・ファイルを、使用可能な最高速のディスク・ドライブに置きます。

- 最高速のディスク・ドライブ上のラージ・ファイルに頻繁に順次アクセスを行うことが予期される場合は、ディスク・アダプター当たりのディスク・ドライブ数を制限してください。
- 可能なら、重要な、大容量のパフォーマンス要件を持つドライブを、高速アダプターに接続します。これらのアダプターには、他のディスク・アダプターにはない、バックツーバック書き込み機能のようなフィーチャーがあります。
- 小型のディスク・ドライブに、大きな頻繁にアクセスされる順次ファイルを保持する論理ボリュームは、物理ボリュームの外部エッジに割り当ててください。 これらのディスクは、その外部セクションにトラック当たりより多くのブロックを持ち、それにより順次パフォーマンスが向上します。
- オリジナル SCSI バスでは、最高の番号が付けられたドライブ (物理ドライブにセットされたとき、数値的に最大の SCSI アドレスを持つもの) が最も高い優先順位を持っています。 その後の指定では、通常、元の指定との互換性を維持しようとします。 したがって、優先順位を最高から最低の順に示すと、次のようになります。7-6-5-4-3-2-1-0-15-14-13-12-11-10-9-8。

  ほとんどの場合、この影響は目立ちませんが、大きい順次ファイルの操作では、下位の番号のドライブはバスへのアクセスから除外されることが知られています。 多くの場合、最も応答時間が重要なデータを保持するディスク・ドライブを、各 SCSI バス上の最高位アドレスに構成するようにしてください。

  lsdev -Cs scsi コマンドは、各 SCSI バス上の現在のアドレス割り当てについて報告します。オリジナル SCSI アダプターの場合、SCSI アドレスは、出力の 4 番目の数値の対の最初の数です。 以下の出力例で、ある 400 GB ディスクは SCSI アドレス 4 に、もう 1 台はアドレス 5 にあり、8 mm 磁気テープ・ドライブはアドレス 1 に、CDROM ドライブはアドレス 3 にあります。

  ```
  cd0    Available 10-80-00-3,0 SCSI Multimedia CD-ROM Drive
  hdisk0 Available 10-80-00-4,0 16 Bit SCSI Disk Drive
  hdisk1 Available 10-80-00-5,0 16 Bit SCSI Disk Drive
  rmt0   Available 10-80-00-1,0 2.3 GB 8mm Tape Drive
  ```
- データベースのように、頻繁に使用され、通常はランダムにアクセスされるラージ・ファイルは、複数の物理ボリュームにまたがって分散させる必要があります。

関連概念:

193 ページの『論理ボリュームおよびディスク入出力のパフォーマンス』
このトピックでは、論理ボリュームとローカル接続ディスク・ドライブのパフォーマンスを取り上げます。

ページング・スペースの配置およびサイズ:

一般的な推奨事項は、ページング・スペースのサイズの合計を、マシンの実メモリーのサイズの少なくとも 2 倍に等しくし、最大メモリー・サイズを 256 MB (512 MB のページング・スペース) にすることです。

注: 256 MB より大きなメモリーの場合には、以下の公式を使用することをお勧めします。

```
total paging space = 512 MB + (memory size - 256 MB) * 1.25
```

ただし、据え置きページ・スペース割り当ての場合、このガイドラインに従うと、実際に必要な量以上のディスク・スペースが占有されることがあります。詳しくは、 168 ページの『ページ・スペース割り当て』を参照してください。

理想的には、ほぼ同サイズの幾つかのページング・スペースが、別々の物理ディスク・ドライブ上にあることです。 追加のページング・スペースを作成することにした場合は、それらを rootvg 内の物理ボリュームより負荷の少ない物理ボリューム上に作成してください。 ページング・スペースのディスク・ブロックを割り当てるとき、VMM は、使用可能なスペースを持つアクティブ・ページング・スペースのそれぞれ

から、順に 4 つのブロックを割り当てます。 システムのブート中は、1 次ページング・スペース (hd6) のみがアクティブです。 結果として、ブート中に割り当てられるページング・スペース・ブロックはすべて 1 次ページング・スペースにあります。 これは、1 次ページング・スペースは 2 次ページング・スペースよりやや大きくなければならないということを意味します。 2 次ページング・スペースは、アルゴリズムが有効に機能するように、すべて同サイズにする必要があります。

lsps -a コマンドは、1 システム上のすべてのページング・スペースについて、現在の使用率レベルのスナップショットを示します。 また、**psdanger()** サブルーチンを使用して、ページング・スペースの使用率がどれほどクリティカルなレベルに近づいているかを判断することもできます。 一例として、以下のプログラムでは、しきい値を超えた場合に警告メッセージが出されるように、**psdanger()** サブルーチンを使用しています。

```
/* psmonitor.c
  Monitors system for paging space low conditions. When the condition is
  detected, writes a message to stderr.
  Usage:    psmonitor [Interval [Count]]
  Default:  psmonitor 1 1000000
*/
#include <stdio.h>
#include <signal.h>
main(int argc,char **argv)
{
  int interval = 1;        /* seconds */
  int count = 1000000;     /* intervals */
  int current;             /* interval */
  int last;                /* check */
  int kill_offset;         /* returned by psdanger() */
  int danger_offset;       /* returned by psdanger() */


  /* are there any parameters at all? */
  if (argc > 1) {
    if ( (interval = atoi(argv[1])) < 1 ) {
      fprintf(stderr,"Usage: psmonitor [ interval [ count ] ]¥n");
      exit(1);
    }
    if (argc > 2) {
      if ( (count = atoi( argv[2])) < 1 ) {
         fprintf(stderr,"Usage: psmonitor [ interval [ count ] ]¥n");
         exit(1);
      }
    }
  }
  last = count -1;
  for(current = 0; current < count; current++) {
    kill_offset = psdanger(SIGKILL); /* check for out of paging space */
    if (kill_offset < 0)
      fprintf(stderr,
        "OUT OF PAGING SPACE! %d blocks beyond SIGKILL threshold.¥n",
        kill_offset*(-1));
    else {
      danger_offset = psdanger(SIGDANGER); /* check for paging space low */
      if (danger_offset < 0) {
        fprintf(stderr,
          "WARNING: paging space low. %d blocks beyond SIGDANGER threshold.¥n",
          danger_offset*(-1));
        fprintf(stderr,
          "                           %d blocks below SIGKILL threshold.¥n",
          kill_offset);
      }
    }
```

```
      if (current < last)
        sleep(interval);
  }
}
```

ディスク・ミラーリングのパフォーマンスへの影響**:**

パフォーマンスの観点からは、ミラーリングにはコストがかかり、書き込み検査 (Write Verify) を指定したミラーリングではさらにコストがかかるので (書き込み当たりのディスク・ローテーションが余分)、書き込み検査およびミラー書き込み整合性の両方を指定したミラーリングは、あらゆるミラーリングの中で最も高くつきます (ディスク・ローテーションに加えてシリンダー 0 へシークのため)。

ミラーリングが使用され、ミラー書き込み整合性 (Mirror Write Consistency) がオンになっている (デフォルトにより) 場合は、ディスクの外部領域にそのコピーを配置することを考慮してください。これは、ミラー書き込み整合性情報が常にシリンダー 0 に書き込まれ、財務の観点からは、書き込みのミラーリングのみ高価になるからです。 **lslv** コマンドは通常、ミラーリングされていない論理ボリュームについてはミラー書き込み整合性がオンになることを表示しますが、COPIES 値が 1 より大きくない限り、実際の処理が発生することはありません。 書き込み検査はデフォルトではオフです。これは、ミラーリングされていない論理ボリュームに対しては書き込み検査が意味を持つ (かつコストがかかる) からです。

受動ミラー書き込み整合性 (MWC) と呼ばれる、ミラー書き込み整合性オプションが使用可能です。ミラー書き込み整合性を確実にするデフォルトのメカニズムが、アクティブ MWC です。 アクティブ MWC により、クラッシュが発生した後のリブート時のリカバリーが速くなります。 ただし、このメリットは書き込みのパフォーマンスを犠牲にして得られるものであり、特に、ランダム書き込みのパフォーマンスが低下します。 アクティブ MWC を使用不可にすれば、この書き込みパフォーマンスの低下はなくなりますが、クラッシュ後のリブートで **syncvg -f** コマンドを使用してボリューム・グループ全体を手作業で同期させてからでないと、ユーザーがそのボリューム・グループにアクセスすることができません。 それには、ボリューム・グループの**自動** `vary-on` を使用不可にしておく必要があります。

受動 MWC を使用可能にした場合は、アクティブ MWC にからむ書き込みパフォーマンスの低下がなくなるだけでなく、パーティションがアクセスされているときに自動的に論理ボリュームの再同期がとられます。 つまり、アドミニストレーターが手作業で論理ボリュームの同期をとる必要がなく、また、**自動** `varyon` を使用不可にする必要もありません。 受動 MWC の欠点としては、すべてのパーティションの同期が取り直されるまでは、読み取り操作が遅くなるということがあります。

論理ボリュームの作成時、または変更時に、SMIT 内でミラー書き込み整合性オプションを選択するかしないかの選択ができます。 この選択オプションが有効になるのは、論理ボリュームのミラーリング (コピー部数 > 1) が行われた場合だけです。

ミラーリングされたストライプ済みの **LV** のパフォーマンスへの影響**:**

論理ボリュームのミラーリングとストライピングによって、RAID 1 のデータ使用可能性と RAID 0 のパフォーマンスがソフトウェアによって完全に結合されます。

論理ボリュームをミラーリングすると同時にストライピングすることはできません。ストライピングされ、ミラーリングされた論理ボリュームを含むボリューム・グループは、AIX のオペレーティング・システムにインポートすることはできません。

通信に関するプリインストールのガイドライン

アダプターの正しい配置方法および各種のパフォーマンス・ガイドラインについては、*PCI Adapter Placement Reference*を参照してください。

279 ページの『TCP および UDP のパフォーマンスのチューニング』および 316 ページの『mbuf プールのパフォーマンスのチューニング』の通信チューニングの推奨事項の要約を参照してください。

POWER4 ベース・システム

POWER4 ベース・サーバーに関連した幾つかのパフォーマンスの問題があります。

関連情報については、 254 ページの『ファイルシステムのパフォーマンス』、 42 ページの『リソース管理』、および IBM Redbooks® 資料の「*The POWER4 Processor Introduction and Tuning Guide*」を参照してください。

POWER4 パフォーマンスの強化

POWER4 マイクロプロセッサーには、幾つかのパフォーマンスの強化が組み込まれています。

- 対称型マルチプロセッシング (SMP) 用に最適化されているため、命令の並列性が良くなっています。
- 命令およびデータのプリフェッチのスケジューリングが改善されているとともに、より効果的な分岐予測メカニズムを採用しています。
- POWER3 マイクロプロセッサーと比較してより高いメモリー帯域幅を備え、より高い周波数で動作するように設計されています。

マイクロプロセッサーの比較

次の表は、種々の IBM マイクロプロセッサーの主な特徴を比較したものです。

表 *1.* プロセッサーの比較

	POWER3	**RS64**	**POWER4**
周波数	450 MHz	750 MHz	> 1 GHz
固定小数点ユニット	3	2	2
浮動小数点ユニット	2	1	2
ロード/ストア・ユニット	2	1	2
ブランチ/その他ユニット	1	1	2
ディスパッチ幅	4	4	5
ブランチ予測	動的	静的	動的
I キャッシュ・サイズ	32 KB	128 KB	64 KB
D キャッシュ・サイズ	128 KB	128 KB	32 KB
L2 キャッシュ・サイズ	1、4、8 MB	2、4、8、16 MB	1.44
L3 キャッシュ・サイズ	N/A	N/A	プロセッサー数についてのスケール
データ・プリフェッチ	はい	いいえ	はい

POWER4 ベース・システムのスケーラビリティー強化

POWER4 ベースのシステムでは、オペレーティング・システムには、ワークロードおよびパフォーマンスの点で、以前のシステムと比較していくつかのスケーラビリティーの利点があります。

ワークロード・スケーラビリティー とは、増大するアプリケーション・ワークロードに対処するための能力のことをいいます。 パフォーマンス・スケーラビリティー とは、より大きなワークロードの要求に答えるためにソフトウェア・リソースが増えたときに、受け入れ可能なパフォーマンス・レベルを維持できることを指します。

最も重要なスケーラビリティーの変更点のいくつかを、以下に示します。

データベースのための固定 (Pinned) 共有メモリー

AIX では、常にメモリー・ページを実メモリー内に維持することができます。 このメカニズムをメモリー固定 と呼びます。

メモリー領域を固定すると、固定されたメモリー領域の裏づけとなるページからのページャーによるページ・スチールが禁止されます。

大容量メモリーのサポート

64 ビット・カーネルがサポートする最大実メモリー・サイズは、ハードウェア・システムによって異なります。

このサイズは、ハードウェア・システムのブート時の実メモリー所要量と 64 ビット・カーネルがサポートする、考えられる入出力構成に基づいています。 64 ビット・カーネルの場合は、ページング・スペース・サイズに最低所要量はありません。

64 ビット・カーネル

AIX オペレーティング・システムは、32-way システムでのスループットに制限を加えていたボトルネックを解消する、64 ビット・カーネルを提供します。

AIX 7.1 の時点で、64 ビット・カーネルは使用可能な唯一のカーネルです。 POWER4 システムは、RS/6000 System p システムのスケーラビリティーの拡大を意図した 64 ビット・カーネル用に最適化されています。 これは、POWER4 システムで実行する 64 ビット・アプリケーション用に最適化されています。 64 ビット・カーネルではより大量の物理メモリーが許可されるため、スケーラビリティーも改善されます。

また JFS2 は AIX 7.1 のデフォルト・ファイルシステムです。 JFS または拡張 JFS のいずれかの使用を選択することができます。 拡張 JFS について詳しくは、『ファイルシステムのパフォーマンス』を参照してください。

32 ビット・カーネル上での 64 ビット・アプリケーション

POWER4 システムの 64 ビット・カーネル上で実行される 64 ビット・アプリケーションのパフォーマンスは、同じハードウェアでの 32 ビット・カーネル上で実行する同じアプリケーションと同等か、より高いパフォーマンスを示すはずです。

64 ビット・カーネルにより、 64 ビット・アプリケーションは、システム・コール・パラメーターを再マップしたりすることなく、サポートされます。 64 ビット・カーネル・アプリケーションは、特に POWER4 システム用に最適化された、さらに拡張化されたコンパイラーを使用します。

64 ビット・カーネル上での 32 ビット・アプリケーション

ほとんどの場合、32 ビット・アプリケーションは性能の低下なしに 64 ビット・カーネルの下で実行できます。

64 ビット・カーネル上の 32 ビット・アプリケーションは、一般的に 32 ビット・コールに比較して、パラメーターの再シェーピングが必要なため、多少、パフォーマンスが落ちます。 このパフォーマンスの低下は、通常、5% 以下です。 例えば、**fork()** コマンドを呼び出すと、非常にオーバーヘッドが大きくなる場合があります。

64 ビット・カーネル、非 POWER4 システム上での 64 ビット・アプリケーション

POWER4 以外のシステムでの 64 ビット・カーネルの下の 64 ビット・アプリケーションのパフォーマンスは、同じハードウェアでの 32 ビット・カーネルの下の同じアプリケーションの場合と比較して低くなる場合があります。

非 POWER4 システムは POWER4 システムへのブリッジを意図したもので、最適な 64 ビット・カーネルのパフォーマンスに必要ないくつかのサポートがありません。

非 POWER4 システム上での 64 ビット・カーネル・エクステンション

POWER4 システムでの 64 ビット・カーネル・エクステンションのパフォーマンスは、同じハードウェアでの対応する 32 ビットのパフォーマンスの場合と同じか、良いはずです。

しかし、非 POWER4 マシンでの 64 ビット・カーネル・エクステンションのパフォーマンスは、同じハードウェアでの 32 ビット・カーネル・エクステンションの場合より低い場合があります。それは、64 ビット・カーネルのパフォーマンスのための最適化が非 POWER4 システムで行われていないためです。

拡張ジャーナル・ファイルシステム

拡張 JFS (JFS2) は、ネイティブ AIX ジャーナル・ファイルシステムの 1 つです。これは AIX 6.1 用のデフォルト・ファイルシステムです。

拡張 JFS の詳細については、 254 ページの『ファイルシステムのパフォーマンス』を参照してください。

マイクロプロセッサーのパフォーマンス

このトピックでは、ランナウェイ・プログラムまたはプロセッサー集中プログラムの検出と、それらがシステム・パフォーマンスに及ぼす悪影響の最小化のための技法について説明します。

マイクロプロセッサーのスケジューリングに慣れていない読者は、 43 ページの『プロセッサー・スケジューラーのパフォーマンス』を参照してください。

マイクロプロセッサーのパフォーマンス・モニター

処理装置は、システムの最高速のコンポーネントの 1 つです。

単一のプログラムが一度に数秒以上の間、マイクロプロセッサーを 100% ビジー (すなわち、0% アイドルで 0% 待ち状態) にしておくことは、比較的まれです。 負荷の重いマルチユーザー・システムでも、すべてのスレッドが待ち状態になって終わる、10 ミリ秒の期間が時々あります。 モニターの結果、マイクロプロセッサーが長期間にわたって 100% ビジーを示している場合、あるプログラムが無限ループに入っている可能性があります。 プログラムが失敗したのではなく、「単に」不経済であるというだけの場合は、そのようなプログラムを区別して、それなりの扱いをする必要があります。

vmstat コマンド

最初に使用するツールは **vmstat** コマンドで、各種システム・リソースと関連するパフォーマンス上の問題に関する情報を迅速に提供してくれます。

vmstat コマンドは、実行キューおよび待機キュー、メモリー、ページング、ディスク、割り込み、システム・コール、コンテキスト切り替え、および CPU アクティビティーにおけるカーネル・スレッドに関する統計情報を報告します。 報告される CPU アクティビティーは、ユーザー・モード、システム・モード、アイドル時間、およびディスク入出力の待ちの分類パーセンテージです。

注: 間隔を指定せずに **vmstat** コマンドが使用される場合、単一のレポートを生成します。 この単一のレポートは、システムが始動した時点からの平均レポートです。 Interval パラメーターを指定した Count パラメーターのみを指定できます。 Interval パラメーターを Count パラメーターと組み合わせないで指定する場合、レポートは継続的に生成されます。

CPU モニターとして、**vmstat** コマンドは、1 レポートごとに 1 行の出力がスクロール時にスキャンしやすく、システムに多数のディスクが接続されている場合もオーバーヘッドが大きくならないので、**iostat** コマンドより優れています。 次の例を見れば、プログラムが暴走しているか、過度の CPU 集中のために、マルチユーザー環境では実行できない状態を確認することができます。

```
# vmstat 2
kthr     memory             page              faults        cpu
----- ----------- ------------------------ ------------ -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 1  0 22478  1677   0   0   0   0    0   0 188 1380 157 57 32  0 10
 1  0 22506  1609   0   0   0   0    0   0 214 1476 186 48 37  0 16
 0  0 22498  1582   0   0   0   0    0   0 248 1470 226 55 36  0  9

 2  0 22534  1465   0   0   0   0    0   0 238  903 239 77 23  0  0
 2  0 22534  1445   0   0   0   0    0   0 209 1142 205 72 28  0  0
 2  0 22534  1426   0   0   0   0    0   0 189 1220 212 74 26  0  0
 3  0 22534  1410   0   0   0   0    0   0 255 1704 268 70 30  0  0
 2  1 22557  1365   0   0   0   0    0   0 383  977 216 72 28  0  0

 2  0 22541  1356   0   0   0   0    0   0 237 1418 209 63 33  0  4
 1  0 22524  1350   0   0   0   0    0   0 241 1348 179 52 32  0 16
 1  0 22546  1293   0   0   0   0    0   0 217 1473 180 51 35  0 14
```

この出力は、使用中のマルチユーザー・システムでタイト・ループに入っているプログラムを実行した場合の影響を示しています。 最初の 3 つのレポート (要約は削除されている) は、システムが 50% から 55% のユーザー、30% から 35% のシステム、および 10% から 15% の入出力待ちで均衡していることを示しています。 プログラムのループが始まると、使用可能な CPU サイクルがすべてそれに消費されます。 ループしているプログラムでは入出力が行われないので、以前は入出力待ちのために使用されていなかったサイクルをすべて吸収してしまう可能性があります。 さらに悪いことに、このプログラムは、有効なプロセスが CPU を解放したときに、いつでもそれを引き継ぐ用意ができているプロセスということになります。 ループしているプログラムは他のすべてのフォアグラウンド・プロセスと同じ優先順位を持っているので、他のプロセスがディスパッチ可能になった場合にも CPU を手放す必要はありません。 プログラムが 10 秒程 (5 つのレポート) 実行した後、**vmstat** コマンドによって報告されるアクティビティーはより正常なパターンに戻ります。

最適の使用方法では、CPU が 100% の時間働きます。これは、CPU を共用する必要のない単一ユーザー・システムの場合は成立します。 一般に、`us + sy` の時間が 90% 未満の場合、単一ユーザー・システムは CPU 制約ではありません。ただし、マルチユーザー・システムの `us + sy` の時間が 80% を超えると、プロセスが実行キューで待っている間に時間を費やしてしまう可能性があります。その結果、応答時間とスループットが低下します。

CPU がボトルネックなのかどうかを調べるには、**vmstat** レポートの 4 つの「`cpu`」欄と 2 つの「`kthr`」(カーネル・スレッド) 欄を検討してください。「`faults`」欄を見るのも役に立ちます。

- **cpu**

  間隔中の CPU 時間使用率の分類パーセンテージ。 「`cpu`」欄の内容は次のとおりです。

  – **us**

    「`us`」欄は、ユーザー・モードで使用された CPU 時間の割合 (%) を示しています。 UNIX プロセスは、ユーザー・モードまたはシステム (カーネル) モードで実行できます。 ユーザー・モードで

は、プロセスはそのアプリケーション・コード内部で実行され、計算の実行、メモリーの管理、あるいは変数の設定にカーネル・リソースを必要としません。

– **sy**

「sy」欄には、CPU がシステム・モードでプロセスを実行していた時間の割合が詳しく示されます。 これには、カーネル・プロセス (**kprocs**) およびカーネル・リソースへのアクセスが必要なその他のプロセスによって消費された CPU リソースが含まれます。 カーネル・リソースを必要とするプロセスは、システム・コールの実行が必要なので、システム・モードへ切り替えてそのリソースを使用可能にします。 例えば、メモリーのマップ・ファイルを使用している場合を除いて、ファイルの読み書きにはそのファイルのオープン、特定ロケーションへのシーク、およびデータの読み書きのためのカーネル・リソースが必要です。

– **id**

「id」欄には、CPU が、ローカル・ディスク入出力を保留せずにアイドル、または待ち状態になっている時間の割合 (%) が示されています。実行に使用できるスレッドがない場合 (実行キューが空)、システムは **wait** というスレッドをディスパッチします。このスレッドは、**idle kproc** としても知られています。 SMP システムでは、各プロセッサーごとに 1 つの **wait** スレッドをディスパッチできます。 **ps** コマンド (**-k** または **-g 0** オプションを指定した) によって生成されたレポートは、これを kproc または wait として示します。 **ps** レポートでこのスレッドの集約時間が大きいことが示された場合、それは、長時間にわたって、他のスレッドが実行可能になっていないか、他に CPU で実行されるのを待っているスレッドがないことを意味します。 したがって、システムはほとんどアイドル および新規タスク待ち 状態でした。

– **wa**

「wa」欄には、ローカル・ディスクおよび NFS マウントのディスクの入出力が保留中で CPU がアイドル だった時間の割合 (%) を詳しく示します。 **wait** の実行中にディスクに対する未解決の入出力がある場合、その時間は入出力待ちに分類されます。プロセスが非同期通信 I/O を使用している場合を除き、ディスクに対する入出力要求では、要求が完了するまで呼び出しプロセスはブロック (またはスリープ) します。プロセスの入出力要求が完了すると、呼び出しプロセスは実行キューに置かれます。 入出力が早く完了していれば、CPU 時間を多く使用できた可能性があります。

wa 値が 25% を超えている場合は、ディスク・サブシステムの平衡が正しく保てないことを示しているか、またはディスク集中のワークロードの結果である可能性があります。

wa に加えられた変更に関する情報は、 194 ページの『入出力待ち時間の報告』を参照してください。

• **kthr**

サンプリング間隔中における、さまざまなキュー内での 1 秒当たり平均カーネル・スレッド数。
「kthr」欄の内容は次のとおりです。

– **r**

実行可能なカーネル・スレッドの平均数。この中には、実行中のスレッドと、CPU 待ちのスレッドが含まれます。 この数が、CPU の数より多いときは、少なくとも 1 つのスレッドが CPU を待っています。CPU 待ちのスレッド数が多いほど、パフォーマンスへの影響が大きくなる傾向にあります。

– **b**

VMM 待機キュー内にある 1 秒当たりの平均カーネル・スレッド数。 この中には、ファイルシステムの入出力を待っているスレッド、またはメモリー・ロード制御のために中断されていたスレッドが含まれます。

プロセスがメモリー・ロード制御のために中断された場合、**vmstat** レポート内の「blocked」欄 (b) は、実行キューではなく、スレッド数の増加を示しています。

- **p**

  vmstat -I の場合、ロー・デバイスへの入出力を待っている 1 秒当たりスレッド数。 ファイルシステムへの入出力を待っているスレッドは、ここに含まれません。

- **faults**

トラップや割り込み率などの、プロセス制御に関する情報。 `faults` 欄の内容は次のとおりです。

- **in**

  その間隔で監視されたデバイス割り込みの 1 秒当たりの数。 追加情報は、 197 ページの『vmstat コマンドによるディスク・パフォーマンスの評価』にあります。

- **sy**

  その間隔中に監視されたシステム・コールの 1 秒当たりの数。 リソースは、明確に定義されたシステム・コールによって、ユーザー・プロセスが使用できます。 このようなコールは、カーネルに対して呼び出しプロセスのための操作を実行し、カーネルとプロセスの間でデータを交換するように指示します。 ワークロードとアプリケーションの種類はさまざまで、さらに多様なコールが異なる機能を実行するので、1 秒当たりのシステム・コール数がどの程度になると多すぎるのかを定義することは不可能です。 ただし一般的には、`sy` 欄の値がユニプロセッサーで 1 秒当たり 10000 コールを超えた場合、さらに調査する必要があります (SMP システムの場合、この数は 1 プロセッサー当たり 1 秒に 10000 コールです)。 この理由の 1 つは、例えば **select()** サブルーチンのような「ポーリング」サブルーチンです。この欄については、通常の `sy` 値のカウントを示す基準となる測定値があると役に立ちます。

- **cs**

  その間隔で監視されたコンテキスト切り替えの 1 秒当たりの数。 物理的 CPU リソースは、それぞれが 10 ミリ秒の論理タイム・スライスに分割されます。 スレッドは、実行のためにスケジュールされているとすると、タイム・スライスが期限切れになるまで、他のスレッドに優先使用されるまで、または自発的に CPU の制御を手放すまで、実行を続けます。他のスレッドが CPU の制御を与えられると、直前のスレッドのコンテキストまたは作業環境を保管して、現行スレッドのコンテキストをロードする必要があります。 オペレーティング・システムには非常に有効なコンテキスト切り替えプロシージャーがあるので、切り替えのたびにリソースを多く消費することはありません。 `cs` がディスク入出力とネットワーク・パケットの速度よりずっと高い場合のように、コンテキストの切り替えが大幅に増加した場合は、さらに調査が必要になります。

iostat コマンド

iostat コマンドは、システムに、ディスク入出力の制約によるパフォーマンス上の問題があるかどうかに関して、最初の印象を得る最も速い方法です

194 ページの『iostat コマンドによるディスク・パフォーマンスの評価』を参照してください。 このツールには、CPU 統計情報のレポート機能もあります。

次の例は、**iostat** コマンドの出力の一部を示しています。 最初のスタンザは、システム起動以来の統計の要約です。

```
# iostat -t 2 6
tty:      tin         tout   avg-cpu:  % user    % sys     % idle    % iowait
          0.0          0.8               8.4      2.6       88.5       0.5
          0.0         80.2               4.5      3.0       92.1       0.5
          0.0         40.5               7.0      4.0       89.0       0.0
          0.0         40.5               9.0      2.5       88.5       0.0
          0.0         40.5               7.5      1.0       91.5       0.0
          0.0         40.5              10.0      3.5       80.5       6.0
```

CPU 統計情報欄 (*% user*、*% sys*、*% idle*、および *% iowait*) は、CPU 使用率の明細を提供します。 この情報は、 **vmstat** コマンドの出力でも、「*us*」、「*sy*」、「*id*」、および「*wa*」というラベルのついた欄に報告されます。 これらの値の詳細については、 112 ページの『vmstat コマンド』を参照してください。さらに、 194 ページの『入出力待ち時間の報告』で説明する、「*%iowait*」に加えられた変更についても注意してください。

関連タスク:

194 ページの『iostat コマンドによるディスク・パフォーマンスの評価』
評価は、システムのワークロードのピーク期間中、または入出力遅延を最小にする必要がある重要なアプリケーションの実行中に、interval パラメーターを指定して **iostat** コマンドを実行することによって開始します。

関連資料:

16 ページの『iostat コマンドを使用した連続パフォーマンス・モニター』
iostat コマンドは、ディスクおよび CPU の使用状況を判別するために便利です。

sar コマンド

sar コマンドは、システムに関する統計データを収集します。

sar コマンドは、システム・パフォーマンスに関する一部の有用なデータの収集に使用できますが、サンプリングの頻度を高くすると、システム負荷が増加して、以前から存在していたパフォーマンス上の問題を悪化させる可能性があります。 しかしアカウンティング・パッケージと比較すれば、**sar** コマンドによる影響は軽微です。 システムは一連のシステム・アクティビティー・カウンターを維持しており、これによってさまざまなアクティビティーを記録すると共に、**sar** コマンドで報告されるデータを提供しています。 **sar** コマンドによってこれらのカウンターが更新または使用されることはありません。この作業は、**sar** コマンドの実行とは関係なく、自動的に行われます。 この作業は、単に **sar** コマンドに指定されたサンプリング率とサンプルの数に基づいて、カウンターのデータを取り出して保管するだけです。

sar コマンドには多数のオプションがあり、キューイング、ページング、TTY、およびその他多数の統計情報を提供します。 **sar** コマンドの 1 つの重要な特徴は、システム全体の (全プロセッサーの包括的な) CPU 統計情報 (これは、割合で表される値については平均値を算出し、それ以外については合計する) か、または各プロセッサーごとの統計情報のいずれかを報告することです。 したがって、このコマンドは SMP システムで特に有効です。

sar コマンドを使用する状況には、次の 3 通りがあります。

リアルタイムのサンプリングと表示**:**

システム統計レポートを収集してすぐに表示する場合は、**sar** コマンドを実行します。

次のコマンドを使用します。

```
# sar -u 2 5

AIX ses12 1 6 000126C5D600    04/08/08

System configuration: lcpu=2  mode=Capped

19:42:43    %usr    %sys    %wio   %idle   physc
19:42:45       0       2       1      97    0.98
19:42:47       0       0       0     100    1.02
19:42:49       0       0       0     100    1.00
19:42:51       0       0       0     100    1.00
19:42:53       0       0       0     100    1.00

Average        0       1       0      99    1.00
```

この例はシングル・ユーザー・ワークステーションでの、CPU 使用率を表示しています。

以前取り込んだデータの表示**:**

-o オプションと **-f** オプション (ユーザー指定のデータ・ファイルへの書き込みおよび、そこからの読み取り) を使用すると、ユーザーのマシンの動作を、2 つの独立したステップで視覚化することができます。この方法では、問題再現期間のリソースの消費が少なくなります。

収集されたバイナリー・ファイルに **sar** コマンドに必要なデータがすべて入っているので、ファイルを転送して、別のマシンを使用してデータを分析することができます。

```
# sar -o /tmp/sar.out 2 5 > /dev/null
```

上記のコマンドは、**sar** コマンドをバックグラウンドで実行し、システム・アクティビティー・データを 2 秒間隔で 5 間隔分収集して、(不定様式の) **sar** データを /tmp/sar.out ファイルに保管します。 標準出力のリダイレクトは、画面出力を避けるために使用されます。

次のコマンドは、ファイルから CPU 情報を抽出し、フォーマット済みレポートを標準出力に出力します。

```
# sar -f/tmp/sar.out

AIX ses12 1 6 000126C5D600    04/08/08

System configuration: lcpu=2  mode=Capped

20:17:00    %usr    %sys    %wio   %idle   physc
20:18:00       0       1       0      99    1.00
20:19:00       0       1       0      99    1.00
20:20:00       0       1       0      99    1.00
20:21:01       0       1       0      99    1.00
20:22:00       0       0       0      99    1.00

Average        0       1       0      99    1.00
```

取り込まれたバイナリー・データ・ファイルには、レポートに必要なすべての情報が入っています。 つまり、可能な **sar** レポートをすべて調べることができます。 さらに、シングル・プロセッサー・システム上の SMP システムのプロセッサー個々の情報を表示することもできます。

cron デーモンによるシステム・アクティビティーのアカウンティング**:**

2 つのシェル・スクリプト (/usr/lib/sa/sa1 および /usr/lib/sa/sa2) は、**cron** デーモンによって実行され、1 日ごとの統計情報とレポートを提供するように構成されています。

sar コマンドは、**sadc** という名前のプロセスを呼び出して、システム・データにアクセスします。 サンプル・スタンザは、 /var/spool/cron/crontabs/adm crontab ファイルに組み込まれて (ただし、コメント化されて)、**cron** デーモンがシェル・スクリプトを実行すべき時期を指定します。

以下の行は、adm ユーザー用に変更された crontab を示しています。 データ収集のコメント文字だけが削除されました。

```
#==================================================================
#      SYSTEM ACTIVITY REPORTS
#  8am-5pm activity reports every 20 mins during weekdays.
#  activity reports every an hour on Saturday and Sunday.
#  6pm-7am activity reports every an hour during weekdays.
#  Daily summary prepared at 18:05.
#==================================================================
0 8-17 * * 1-5 /usr/lib/sa/sa1 1200 3 &
0 * * * 0,6 /usr/lib/sa/sa1 &
0 18-7 * * 1-5 /usr/lib/sa/sa1 &
5 18 * * 1-5 /usr/lib/sa/sa2 -s 8:00 -e 18:01 -i 3600 -ubcwyaqvm &
#==================================================================
```

データをこのような方法で収集すると、ある期間のシステム使用の特徴を明確にして、ピーク使用時間を判別する場合に役に立ちます。

便利なマイクロプロセッサー・オプション*:*

sar コマンドには、多数の便利なマイクロプロセッサー関連オプションがあります。

最も便利なオプションは、次のとおりです。

- **sar -P**

  -P オプションは、指定されたプロセッサーのプロセッサー別の統計情報を報告します。 **ALL** キーワードを指定すると、個々のプロセッサーごとの統計情報と全プロセッサーの平均が報告されます。 ワークロード区画環境の内部で **-P ALL** が使用されるとき、システム全体の統計情報に加えて RSET 全体の統計情報が表示され、RSET に所属するプロセッサーの前にアスタリスク (*) シンボルが付けられます。 RSET 全体の統計情報は、ワークロード区画環境が RSET に関連付けられている場合にのみ表示されます。報告すべき統計情報を指定するフラグの中で、**-a**、**-c**、**-m**、**-u**、および **-w** のフラグだけが、**-P** フラグで意味を持ちます。

  以下の例は、マイクロプロセッサー制約のプログラムがプロセッサー番号 0 で実行した場合のプロセッサーごとの統計です。

  ```
  # sar -P ALL 2 2

  AIX tooltime2 1 6 00CA52594C00    04/02/08
  System configuration: lcpu=4  mode=Capped
  05:23:08 cpu    %usr    %sys    %wio   %idle   physc
  05:23:11  0        8      92       0       0    1.00
            1        0      51       0      49    0.00
            2        0       0       0     100    0.51
            3        0       0       0     100    0.48
            -        4      46       0      50    1.99
  05:23:13  0       10      89       0       0    1.00
            1        0       7       0      93    0.00
            2        0       3       2      95    0.51
            3        0       0       0     100    0.49
            -        5      45       0      49    2.00

  Average   0        9      91       0       0    1.00
            1        0      12       0      88    0.00
  ```

```
          2        0       2       1      98    0.51
          3        0       0       0     100    0.48
          -        5      46       0      49    1.99
```

すべてのスタンザの最後の行は、「cpu」欄がダッシュ (-) で始まっており、全プロセッサーの平均を示しています。 平均 (-) の行が表示されるのは、**-P ALL** オプションを使用した場合だけです。 プロセッサーを指定した場合、この行は削除されます。最後のスタンザは、ラベルにタイム・スタンプの代わりに「Average」という語が使用されていますが、すべてのスタンザにわたるプロセッサー固有の行の平均を保持しています。

次の例は、この時間における **vmstat** 出力を示しています。

```
System configuration: lcpu=4 mem=44570MB
kthr    memory              page              faults        cpu
----- ----------- ------------------------ ------------ -----------
 r  b   avm   fr       re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 2  0 860494 6020610   0   0   0   0    0   0  16 14061 409  5 45 49  0
 2  0 860564 6020540   0   0   0   0    0   0   4 14125 400  5 45 50  0
 1  0 860669 6020435   0   0   0   0    0   0   3 14042 388  5 46 49  0
 2  0 860769 6020335   0   0   0   0    0   0   3 13912 398  5 45 50  0
```

最初の番号付きの行は、システム起動以来の要約です。 2 番目の行は **sar** コマンドの開始を反映しており、3 番目の行で、レポートを比較できるようになります。 **vmstat** コマンドは、全プロセッサーの平均マイクロプロセッサー使用率だけを表示することができます。これは、**sar** コマンドのマイクロプロセッサー使用率の出力のダッシュ (-) の付いた行と同等です。

WPAR 環境の内部で実行されるとき、同じコマンドは次の出力を作成します。

```
AIX wpar1 1 6 00CBA6FE4C00    04/01/08
wpar1 configuration: lcpu=2  memlim=204MB  cpulim=0.06  rset=Regular
05:23:08 cpu    %usr    %sys    %wio   %idle   physc
05:23:11 *0        8      92       0       0    1.00
         *1        0      51       0      49    0.00
          2        0       0       0     100    0.51
          3        0       0       0     100    0.48
          R        4      71       0      24    1.00
          -        4      46       0      50    1.99
05:23:13 *0       10      89       0       0    1.00
         *1        0       7       0      93    0.00
          2        0       3       2      95    0.51
          3        0       0       0     100    0.49
          R        5      48       0      46    1.00
          -        5      45       0      49    2.00

Average  *0        9      91       0       0    1.00
         *1        0      12       0      88    0.00
          2        0       2       1      98    0.51
          3        0       0       0     100    0.48
          R        4      51       0      44    1.00
          -        5      46       0      49    1.99
```

WPAR は関連付けられた RSET レジストリーを持ちます。 プロセッサー 0 および 1 は RSET に取り付けられます。 R 行は、WPAR に関連付けられた RSET による使用を示します。 RSET 内に存在するプロセッサーの前にアスタリスク (*) シンボルが付けられます。

- **sar –P** RST は、RSET 内に存在するプロセッサーの使用メトリックを表示するために使用されます。 WPAR 環境に関連付けられた RSET が存在しない場合、すべてのプロセッサー・メトリックが表示されます。

次の例では、WPAR 環境内部で実行された **sar –P** RST を示します。

```
AIX wpar1 1 6 00CBA6FE4C00    04/01/08

wpar1 configuration: lcpu=2  memlim=204MB  cpulim=0.06  rset=Regular

05:02:57 cpu    %usr    %sys    %wio   %idle   physc
05:02:59  0       20      80       0       0    1.00
          1       10       0       0      90    0.00
          R       15      40       0      45    1.00
05:03:01  0       20      80       0       0    1.00
          1        8       0       0      92    0.00
          R       14      40       0      46    1.00

Average   0       20      80       0       0    1.00
          1        9       0       0      91    0.00
          R       14      40       0      46    1.00
```

- **sar -u**

  これはマイクロプロセッサー使用率を表示します。 他にフラグが指定されていない場合は、これがデフォルトです。 表示される情報は、**vmstat** または **iostat** コマンドのマイクロプロセッサー統計情報と同じです。

  次の例の間に、コピー・コマンドが開始されました。

```
# sar -u 3 3
AIX wpar1 1 6 00CBA6FE4C00    04/01/08
wpar1 configuration: lcpu=2  memlim=204MB  cpulim=0.06  rset=Regular
05:02:57 cpu    %usr    %sys    %wio   %idle   physc
05:02:59  0       20      80       0       0    1.00
          1       10       0       0      90    0.00
          R       15      40       0      45    1.00
05:03:01  0       20      80       0       0    1.00
          1        8       0       0      92    0.00
          R       14      40       0      46    1.00

Average   0       20      80       0       0    1.00
          1        9       0       0      91    0.00
          R       14      40       0      46    1.00
```

  ワークロード・パーティションの内部で実行されるとき、同じコマンドは次の出力を作成します。

```
AIX wpar1 1 6 00CBA6FE4C00    04/01/08

wpar1 configuration: lcpu=2  memlim=204MB  cpulim=0.06  rset=Regular

05:07:16    %usr    %sys    %wio   %idle   physc   %resc
05:07:19      17      83       -       -    0.11   181.6
05:07:22      19      81       -       -    0.08   133.5
05:07:26      16      84       -       -    0.10   173.4

Average       17      83       -       -    0.10   164.3
```

  これは、プロセッサー・リソース制限に強制させたワークロード・パーティションの %resc 情報を表示します。 メトリックはワークロード・パーティションにより消費されたプロセッサー・リソースのパーセンテージを示します。

- **sar -c**

  -c オプションは、システム・コール率を表示します。

```
# sar -c 1 3
19:28:25 scall/s sread/s swrit/s  fork/s  exec/s rchar/s wchar/s
19:28:26     134      36       1    0.00    0.00 2691306    1517
19:28:27      46      34       1    0.00    0.00 2716922    1531
```

```
19:28:28      46      34       1    0.00    0.00 2716922    1531

Average       75      35       1    0.00    0.00 2708329    1527
```

vmstat コマンドもシステム・コール率を表示しますが、**sar** コマンドは、システム・コールが **read()**、**write()**、**fork()**、**exec()**、およびその他かどうかも表示します。 「fork/s」欄には特に注意が必要です。 この欄の値が高い場合、アカウンティング・ユーティリティー、**trace** コマンド、または **tprof** コマンドを使用した、さらに詳しい調査が必要になる可能性があります。

- **sar -q**

  -q オプションは、実行キュー・サイズとスワップ・キュー・サイズを表示します。

  ```
  # sar -q 5 3

  19:31:42 runq-sz %runocc swpq-sz %swpocc
  19:31:47     1.0     100     1.0     100
  19:31:52     2.0     100     1.0     100
  19:31:57     1.0     100     1.0     100

  Average      1.3      95     1.0      95
  ```

  runq-sz
  1 秒当たり実行可能な平均スレッド数、および実行キューが使用されていた時間のパーセンテージ (% フィールドはエラーの可能性がある)。

  swpq-sz
  VMM 待機キュー内の平均スレッド数、およびスワップ・キューが使用されていた時間のパーセンテージ。 (% フィールドはエラーの可能性がある。)

  -q オプションは、ユーザーの実行しているジョブが多すぎる (runq-sz) か、またはページング・ボトルネックの可能性があるかどうかを示すことができます。 大量のトランザクションが発生するシステム (例えば、エンタープライズ・リソース・プランニング (ERP)) では、それぞれのトランザクションはマイクロプロセッサー時間の使用が少ないため、実行キュー内に数百も入る可能性があります。 ページングが問題である場合、**vmstat** コマンドを実行します。 入出力待ちが多い場合は、かなりのディスク・アクティビティーの競合、またはメモリーが不十分なための大量のページングが存在することを示しています。

リソース・セットの使用:

リソース・セットを使用するためのベスト・プラクティスについては、以下のガイドラインに従ってください。

推奨事項

1. リソース・セットがサブコアであるが、各コアからの複数のプロセッサーを含んでいる場合は、最良の結果を得るために、コアに組み込まれている各プロセッサーにギャップが存在しないように、プライマリー・スレッド、および後続の連続同時マルチスレッド化 (SMT) スレッド (論理プロセッサー) を組み込んでください。
2. リソース・セットには、表される各コアからの同じ数のプロセッサーが含まれます。

ガイドライン

以下の記述は、リソース・セットの使用時に当てはまります。

- リソース・セットは、複数のスケジューラー・リソース割り振りドメイン (SRAD) からのコアを保持することができ、各 SRAD からの同じ数のコアを保持する必要はありません。

- リソース・セットは、表される各コアからの 1 つのプロセッサーを保持することができ、そのプロセッサーはプライマリー SMT スレッドである必要はありません。
- リソース・セット用のロード・バランシングが有効になっていると、VPM などのプロセッサー・フォールディング・サブシステムは、リソース・セットで必要とされるコアに基づくフォールディング決定を行います。ほとんどのリソース・セットが接続される作業を持つコアには、フォールディングまたはアンフォールディングされるコアが決定されるときに優先順位が付与されます。

  注**:** 静的省電力モードが有効になっている場合、VPM は、リソース・セット用のロード・バランシングが有効になっていても、エネルギー認識コア選択を行います。

xmperf プログラム

xmperf プログラムを使用すると、CPU 使用状況が移動スカイ・ライン図表の形で表示されます。

マイクロプロセッサー使用率測定のための time コマンドの使用法

単一プログラムとそれに同期している子のパフォーマンス特性を理解するには、**time** コマンドを使用します。

time コマンドは、プログラムの開始から終了までの経過時間である、*real (*実*)* 時間を報告します。 これに加えて、プログラムが使用したマイクロプロセッサー時間も報告します。 マイクロプロセッサー時間は user と sys に分けられます。 user 値は、プログラム自体とプログラムが呼び出したライブラリー・サブルーチンが使用した時間です。 sys 値は、プログラムによって起動されたシステム・コールが使用した時間です。

user と sys の合計は、プログラム実行による直接マイクロプロセッサーの合計コストです。 このコストには、プログラムのために実行するということであっても、実際にはそのスレッドで実行しないカーネルの部分のマイクロプロセッサー・コストは含まれません。例えば、プログラム開始時にフリー・リストから取ったページ・フレームを置き換えるためのページ・フレームをスチールするコストは、そのプログラムのマイクロプロセッサー使用量の一部としては報告されません。

ユニプロセッサーでは、real 時間と合計マイクロプロセッサー時間との差、すなわち、

```
real - (user + sys)
```

は、プログラムを遅らせる可能性のあるすべての要因に、プログラム自体の属性が付いていないコストを加えたものです。 SMP では、近似値は次のようになります。

```
real * number_of_processors - (user + sys)
```

要因は、サイズを減少させるおおよその順序に並べると、次のようになります。

- プログラムのテキストおよびデータの読み込みに必要な入出力
- プログラムで使用するための実メモリーの獲得に必要な入出力
- その他のプログラムが消費するマイクロプロセッサー時間
- オペレーティング・システムが消費するマイクロプロセッサー時間

次の例では、前のセクションで使用されたプログラムは、より迅速に実行するために、**-O3** でコンパイルされています。 プログラムの実行に必要な実経過時間とそのユーザーとシステムのマイクロプロセッサー時間の合計の間には、非常に小さい差しかありません。 このプログラムは、おそらくシステム内のほかのプログラムを犠牲にして、いつでも欲しいものを手に入れています。

```
# time looper
real    0m3.58s
user    0m3.16s
sys     0m0.04s
```

次の例では、nice の値に 10 を加えて、より低い優先順位でプログラムを実行します。 この場合は実行時間が 2 倍近くになりますが、他のプログラムも作業を行う機会を得られます。

```
# time nice -n 10 looper
real    0m6.54s
user    0m3.17s
sys     0m0.03s
```

上記の例では、**time** コマンド中に **nice** コマンドを入れています。その逆ではないことに注意してください。 前の例と逆に入力すると、次のようになります。

```
# nice -n 10 time looper
```

この場合は、**ksh** シェルに組み込まれている、これまで使用してきた **time** コマンドでなく、精度の低いレポートを作成する、異なる **time** コマンド (**/usr/bin/time**) になります。 **time** コマンドが最初に来た場合は、完全修飾名の **/usr/bin/time** を指定しなければ、組み込みバージョンになります。 別のコマンドから **time** コマンドを呼び出した場合は、**/usr/bin/time** が得られます。

time および timex コマンドの考慮事項

time または **timex** コマンドのいずれかを使用するときは、以下の幾つかの事柄に注意してください。

以下の要因があります。

- /usr/bin/time および /usr/bin/timex コマンドの使用は、お勧めできません。 可能な場合は、Korn または C シェルの time サブコマンドを使用してください。
- **timex -s** コマンドは、**sar** コマンドを使用して追加の統計情報を獲得します。 これは、**sar** コマンドが強引に割り込みを行うからであり、**timex -s** コマンドも同様です。 特に短い実行の場合は、**timex -s** コマンドの報告したデータが、モニターされていないシステムでのプログラムの動作を正確に反映していない場合があります。
- システム・クロックのティックの長さ (10 ミリ秒) と、スケジューラーが CPU 時間の使用がスレッドによるものかを決める際に使用する規則のために、**time** コマンドの結果は完全に正確と言えるものではありません。 時間のサンプリングを行うので、継続的な実行と実行の間に一定量の避けがたい変動が存在します。 この変動は、クロックの目盛りに関するものです。 プログラムの実行時間が短くなれば、報告される結果のパーセンテージとしての変動は大きくなります (449 ページの『プロセッサー・タイマーのアクセス』を参照)。
- **time** または **timex** コマンドを使用して (/usr/bin からまたは組み込みシェルの **time** 関数によって)、パイプによって接続されたコマンドのシーケンスをコマンド・ラインから入力して、ユーザー時間またはシステム時間を測定するのはお勧めできません。 起こり得る問題の 1 つは、構文の手落ちによって、**time** コマンドが、ユーザー・エラーを表示しないままで 1 つのコマンドだけを測定する可能性です。構文が技術的に正しくても、ユーザーが意図した答えが出てこない場合がありえます。
- SMP 環境では、**time** コマンド構文は変更されていませんが、出力が新しい意味を持つようになりました。

  SMP では、リアルタイムすなわち経過時間がプロセスのユーザー時間より小さくなる場合があります。ユーザー時間は、現在は全プロセッサー上のスレッドまたはプロセスによって費やされた時間すべての合計です。

プロセスに 4 つのスレッドがある場合、それをユニプロセッサー (UP) システムで実行すると、実時間がユーザー時間より大きくなります。

```
# time 4threadedprog
real    0m11.70s
user    0m11.09s
sys     0m0.08s
```

それを 4-way SMP システムで実行すると、リアルタイムはユーザー時間のだいたい 1/4 を示すに過ぎない場合があります。 次の出力は、マルチスレッド化プロセスのワークロードを幾つかのプロセッサーに分散して、その実行時間を改善させたことを示しています。 その結果、システムのスループットは増加しました。

```
# time 4threadedprog
real    0m3.40s
user    0m9.81s
sys     0m0.09s
```

マイクロプロセッサー集中プログラム識別

マイクロプロセッサー使用率が高いプロセスを突き止めるには、2 つの標準ツール、**ps** コマンドおよび **acctcom** コマンドがあります。

他に使用するツールは **topas** モニターで、これについては 18 ページの『topas コマンドを使用した連続システム・パフォーマンス・モニター』に説明があります。

ps コマンドの使用法

ps コマンドは、システム上で実行中のプログラムと、プログラムで使用しているリソースの識別のための柔軟なツールです。 システム上のプロセスに関する統計と状況情報 (プロセス ID、スレッド ID、入出力アクティビティー、CPU、メモリー使用率など) が表示されます。

このセクションでは、CPU に関係のあるオプションと出力フィールドについてのみ説明します。

可能な **ps** 出力欄は 3 つあり、それぞれ異なる方式で CPU 使用量を報告します。

欄　　値:

C　　そのプロセスで一番最近使用された CPU 時間 (クロックの目盛り単位)。

TIME　プロセスが始動以来使用した合計 CPU 時間 (分と秒単位)。

%CPU
　　プロセスが始動以来使用した合計 CPU 時間を、プロセス始動以来の経過時間で割った値。 この時間はプログラムの CPU 依存度を測る尺度です。

CPU 集中

次のシェル・スクリプト、

```
# ps -ef | egrep -v "STIME|$LOGNAME" | sort +3 -r | head -n 15
```

は、システム内で最近 CPU を最も多く使用した CPU 集中ユーザー・プロセスを示すツールです (ヘッダー行は分かりやすくするためにあらためて付け加えてあります)。

```
    UID   PID  PPID   C    STIME    TTY  TIME CMD
   mary 45742 54702 120 15:19:05 pts/29  0:02 ./looper
   root 52122     1  11 15:32:33 pts/31 58:39 xhogger
   root  4250     1   3 15:32:33 pts/31 26:03 xmconsole allcon
   root 38812  4250   1 15:32:34 pts/31  8:58 xmconstats 0 3 30
   root 27036  6864   1 15:18:35      -  0:00 rlogind
```

```
    root 47418 25926   0 17:04:26      -  0:00 coelogin <d29dbms:0>
    bick 37652 43538   0 16:58:40  pts/4  0:00 /bin/ksh
    bick 43538     1   0 16:58:38      -  0:07 aixterm
     luc 60062 27036   0 15:18:35 pts/18  0:00 -ksh
```

C 欄は最近使用された CPU を示します。ループするプログラムのプロセスは、リストの先頭に来ます。スケジューラーが 120 でカウントを停止するので、ループするプロセスの CPU 使用量を C の値により最小化できます。マルチスレッド・プロセスの場合、このフィールドはそのプロセス内のすべてのスレッドについてリストされた CP の和を示します。

次の例は、すべてのスレッドが 1 つの無限ループ・プログラム内にある単純な 5 スレッドのプログラムを示しています。

```
ps -lmo THREAD -p 8060956
    USER     PID    PPID       TID ST  CP PRI SC    WCHAN        F     TT BND COMMAND
    root 8060956 6815882         - A  720 120  0        -   200001  pts/0   - ./a.out
       -       -       -   8716483 R  120 120  0        -   400000      -   - -
       -       -       -  17105017 R  120 120  0        -   400000      -   - -
       -       -       -  24182849 R  120 120  0        -   400000      -   - -
       -       -       -  24510589 R  120 120  0        -   400000      -   - -
       -       -       -  30277829 R  120 120  0        -   400000      -   - -
       -       -       -  35913767 R  120 120  0        -   400000      -   - -
```

「CP」欄の値 720 は、この値の下にリストされた個別スレッドの和、すなわち (5 * 120) + (120) です。

CPU 時間比率

ps コマンドは、周期的に実行されて、「TIME」欄の下に CPU 時間を表示し、「%CPU」欄の下に CPU 時間のリアルタイムに対する比率を表示します。使用率が最高のプロセスを探してください。 **au** オプションと **v** オプションは、ユーザー・プロセスに関して類似した情報を表示します。 **aux** オプションと **vg** オプションは、ユーザー・プロセスとシステム・プロセスの両方を表示します。

次の例は、4-way SMP システムからとったものです。

```
# ps au
USER       PID %CPU %MEM   SZ  RSS    TTY STAT    STIME TIME COMMAND
root     19048 24.6  0.0   28   44  pts/1 A    13:53:00 2:16 /tmp/cpubound
root     19388  0.0  0.0  372  460  pts/1 A      Feb 20 0:02 -ksh
root     15348  0.0  0.0  372  460  pts/4 A      Feb 20 0:01 -ksh
root     20418  0.0  0.0  368  452  pts/3 A      Feb 20 0:01 -ksh
root     16178  0.0  0.0  292  364      0 A      Feb 19 0:00 /usr/sbin/getty
root     16780  0.0  0.0  364  392  pts/2 A      Feb 19 0:00 -ksh
root     18516  0.0  0.0  360  412  pts/0 A      Feb 20 0:00 -ksh
root     15746  0.0  0.0  212  268  pts/1 A    13:55:18 0:00 ps au
```

「%CPU」は、プロセスが始動されて以来、そのプロセスに割り当てられた CPU 時間のパーセンテージです。 次の式で算出します。

```
(process CPU time / process duration) * 100
```

次のような 2 つのプロセスを想像してください。最初のプロセスは始動して 5 秒間実行しますが、終了しません。その後に 2 番目のプロセスが始動して 5 秒間実行しますが、終了しません。この場合、**ps** コマンドは最初のプロセスに 50% の「%CPU」(10 秒の経過時間に対して 5 秒の CPU) を示し、2 番目のプロセスには 100% (5 秒の経過時間に対して 5 秒の CPU) を示します。

SMP の場合、この値はシステム上で使用可能な CPU の数で割ります。 もう一度前の例を見てください。この例は 4-Way プロセッサー・システムで実行していて、**cpubound** の処理の %CPU 値が絶対に 25 を超えないのはそのためだと分かります。 **cpubound** プロセスは 1 つのプロセッサーを 100% 使用して

いますが、「%CPU」は使用可能な CPU の数で割ります。

THREAD オプション

ps コマンドは、**ps -mo THREAD** コマンドを使用することによって、スレッドおよびスレッドかプロセスがバインドされている CPU を表示することができます。 次に例を示します。

```
# ps -mo THREAD
USER PID   PPID  TID   ST CP PRI SC WCHAN F      TT    BND COMMAND
root 20918 20660 -     A  0  60  1  -     240001 pts/1 -   -ksh
-    -     -     20005 S  0  60  1  -     400    -     -   -
```

「TID」欄はスレッド ID を、「BND」欄はプロセッサーにバインドされたプロセスおよびスレッドを示しています。

kproc (AIX オペレーティング・システム バージョン 4 では PID が 516) が CPU 時間を使用しているのは正常です。 タイム・スライス中に実行できるスレッドがない場合、スケジューラーがそのタイム・スライスの CPU 時間をこのカーネル・プロセス (kproc) に割り当てます。これはアイドル または待ち kproc と呼ばれています。 SMP システムでは、プロセッサーごとにアイドルの kproc が 1 つずつあります。

ps コマンドについて詳しくは、 コマンド・リファレンス のを参照してください。

acctcom コマンドの使用法

acctcom コマンドは、アカウンティング・システムが活動化されている場合は、CPU 使用率に関するヒストリー・データを表示します。

アカウンティング・システムを始動すると、システムに無視できないほどのオーバーヘッドが生じるため、どうしても必要な場合のみアカウンティングを活動化してください。 アカウンティング・システムを活動化するには、次の方法に従ってください。

1. 空のアカウンティング・ファイルを作成します。

   ```
   # touch acctfile
   ```

2. アカウンティングをオンにします。

   ```
   # /usr/sbin/acct/accton acctfile
   ```

3. 少しの間アカウンティングを実行させてから、オフにします。

   ```
   # /usr/sbin/acct/accton
   ```

4. アカウンティングで取り込まれたものを、次のように表示します。

   ```
   # /usr/sbin/acct/acctcom acctfile
   COMMAND                      START    END          REAL      CPU     MEAN
   NAME       USER     TTYNAME  TIME     TIME       (SECS)   (SECS)  SIZE(K)
   #accton    root     pts/2   19:57:18 19:57:18     0.02     0.02   184.00
   #ps        root     pts/2   19:57:19 19:57:19     0.19     0.17    35.00
   #ls        root     pts/2   19:57:20 19:57:20     0.09     0.03   109.00
   #ps        root     pts/2   19:57:22 19:57:22     0.19     0.17    34.00
   #accton    root     pts/2   20:04:17 20:04:17     0.00     0.00     0.00
   #who       root     pts/2   20:04:19 20:04:19     0.02     0.02     0.00
   ```

同じファイルを再利用する場合は、**accton** プロセス (これは、最初にアカウンティングをオフにするときに使用されました) を探せば、新しいプロセスが始動されたときは分かります。

カーネル・スレッドのマイクロプロセッサー使用率測定のための pprof コマンドの使用法

pprof コマンドは、ある間隔内の、実行中のすべてのカーネル・スレッドのマイクロプロセッサー使用率を、トレース・ユーティリティーを使用して報告します。

ロー・プロセス情報は、pprof.flow に保管され、5 つのレポートが生成されます。 フラグが指定されていない場合は、すべてのレポートが生成されます。

pprof プログラムがインストールされていて、使用可能かどうかを判別するには、次のコマンドを実行します。

```
# lslpp -lI bos.perf.tools
```

レポートには以下のタイプがあります。

pprof.cpu
実マイクロプロセッサー時間でソートされたすべてのカーネル・レベルのスレッドをリストします。 内容: プロセス名、プロセス ID、親プロセス ID、始動時および終了時のプロセス状態、スレッド ID、親スレッド ID、実 CPU 時間、始動時刻、停止時刻、停止 - 始動。

pprof.famcpu
すべてのファミリー (共通の祖先を持つプロセス) に関する情報をリストします。ファミリーのプロセス名とプロセス ID は必ずしも祖先ではありません。内容: 始動時刻、プロセス名、プロセス ID、スレッドの数、合計 CPU 時間。

pprof.famind
すべてのプロセスを、ファミリー (共通の祖先を持つプロセス) ごとにグループ分けしてリストします。子プロセス名は親に対してインデントされています。 内容: 始動時刻、停止時刻、実 CPU 時間、プロセス ID、親プロセス ID、スレッド ID、親スレッド ID、始動時および終了時のプロセス状態、レベル、プロセス名。

pprof.namecpu
カーネル・スレッドの各タイプ (同じ名前を持つすべての実行可能スレッド) に関する情報をリストします。 内容: プロセス名、スレッドの数、CPU 時間、合計 CPU 時間のパーセンテージ。

pprof.start
pprof コマンドの間隔中にディスパッチされたすべてのカーネル・スレッドを、始動時刻でソートしてリストします。 内容: プロセス名、プロセス ID、親プロセス ID、始動および終了時のプロセス状態、スレッド ID、親スレッド ID、実 CPU 時間、始動時刻、停止時刻、停止 - 始動。

次に示すのは、**tthreads32** プログラムを実行することによって生成されたサンプル pprof.namecpu ファイルで、このプログラムは 4 つのスレッドに fork し、それがさらにそれぞれのプロセスに fork しています。 次に、これらのプロセスが、幾つかの **ksh** プログラムと **sleep** プログラムを実行します。

```
                Pprof  PROCESS  NAME  Report

                Sorted  by  CPU  Time

                From: Thu Oct 19 17:53:07 2000
                To:   Thu Oct 19 17:53:22 2000


      Pname #ofThreads  CPU_Time        %
   ======== ==========  ========  ========
 tthreads32         13     0.116    37.935
         sh          8     0.092    30.087
       Idle          2     0.055    17.987
```

```
           ksh         12     0.026     8.503
         trace          3     0.007     2.289
          java          3     0.006     1.962
         kproc          5     0.004     1.308
       xmservd          1     0.000     0.000
       trcstop          1     0.000     0.000
       swapper          1     0.000     0.000
           gil          1     0.000     0.000
            ls          4     0.000     0.000
         sleep          9     0.000     0.000
            ps          4     0.000     0.000
       syslogd          1     0.000     0.000
          nfsd          2     0.000     0.000
               ==========  ========  ========
                     70     0.306   100.000
```

対応する pprof.cpu は次のとおりです。

```
                Pprof CPU Report

                Sorted  by  Actual CPU  Time

                From: Thu Oct 19 17:53:07 2000
                To:   Thu Oct 19 17:53:22 2000


      E = Exec'd      F = Forked
      X = Exited      A = Alive (when traced started or stopped)
      C = Thread Created

            Pname      PID     PPID  BE      TID     PTID  ACC_time  STT_time  STP_time   STP-STT
            =====    =====    ===== ===    =====    =====  ========  ========  ========  ========
             Idle      774        0  AA      775        0     0.052     0.000     0.154     0.154
       tthreads32     5490    11982  EX    18161    22435     0.040     0.027     0.154     0.126
               sh    11396     5490  EE    21917     5093     0.035     0.082     0.154     0.072
               sh    14106     5490  EE    16999    18867     0.028     0.111     0.154     0.043
               sh    13792     5490  EE    20777    18179     0.028     0.086     0.154     0.068
              ksh     5490    11982  FE    18161    22435     0.016     0.010     0.027     0.017
       tthreads32     5490    11982  CX     5093    18161     0.011     0.056     0.154     0.098
       tthreads32     5490    11982  CX    18179    18161     0.010     0.054     0.154     0.099
       tthreads32    14506     5490  FE    17239    10133     0.010     0.128     0.143     0.015
              ksh    11982    13258  AA    22435        0     0.010     0.005     0.154     0.149
       tthreads32    13792     5490  FE    20777    18179     0.010     0.059     0.086     0.027
       tthreads32     5490    11982  CX    18867    18161     0.010     0.057     0.154     0.097
       tthreads32    11396     5490  FE    21917     5093     0.009     0.069     0.082     0.013
       tthreads32     5490    11982  CX    10133    18161     0.008     0.123     0.154     0.030
       tthreads32    14106     5490  FE    16999    18867     0.008     0.088     0.111     0.023
            trace     5488    11982  AX    18159        0     0.006     0.001     0.005     0.003
            kproc     1548        0  AA     2065        0     0.004     0.071     0.154     0.082
             Idle      516        0  AA      517        0     0.003     0.059     0.154     0.095
             java    11612    11106  AA    14965        0     0.003     0.010     0.154     0.144
             java    11612    11106  AA    14707        0     0.003     0.010     0.154     0.144
            trace    12544     5488  AA    20507        0     0.001     0.000     0.001     0.001
               sh    14506     5490  EE    17239    10133     0.001     0.143     0.154     0.011
            trace    12544     5488  CA    19297    20507     0.000     0.001     0.154     0.153
              ksh     4930     2678  AA     5963        0     0.000     0.154     0.154     0.000
            kproc     6478        0  AA     3133        0     0.000     0.154     0.154     0.000
               ps    14108     5490  EX    17001    18867     0.000     0.154     0.154     0.000
       tthreads32    13794     5490  FE    20779    18179     0.000     0.154     0.154     0.000
               sh    13794     5490  EE    20779    18179     0.000     0.154     0.154     0.000
               ps    13794     5490  EX    20779    18179     0.000     0.154     0.154     0.000
               sh    14108     5490  EE    17001    18867     0.000     0.154     0.154     0.000
       tthreads32    14108     5490  FE    17001    18867     0.000     0.154     0.154     0.000
               ls    13792     5490  EX    20777    18179     0.000     0.154     0.154     0.000
:
:
:
```

emstat ツールによる命令エミュレーションの検出

古いバイナリーとの互換性を保つために、AIX のカーネルには、特定チップのアーキテクチャーに含まれない命令のサポートを提供するエミュレーション・ルーチンが含まれています。 サポートされていない命令を実行しようとすると、無効な命令の例外割り込みになります。 カーネルは、無効な命令をデコードして、それがサポートされていない命令であれば、その命令を機能的にエミュレートするエミュレーション・ルーチンを実行します。

サポートされていない命令の実行頻度とそれらの命令のエミュレーション・パスの長さによっては、エミュレーションを行うと、カーネルのコンテキスト切り替えと命令のエミュレーション・オーバーヘッドによるパフォーマンス低下につながる可能性があります。 エミュレーションの割合が非常に小さくても、パフォーマンスの差は非常に大きくなる場合もあります。 次の表は、幾つかのサポートされていない命令のおよその命令パス長を示しています。

命令	使用する言語	見積パス長さ (命令)
abs	アセンブラー	117
doz	アセンブラー	120
mul	アセンブラー	127
rlmi	C	425
sle	C	447
clf	C	542
div	C	1079

すべてのプラットフォームに共通ではない命令は、再コンパイルは高水準ソース・コードのみで有効なので、アセンブラーで作成されたコードから除去する必要があります。 アセンブラーで作成されたルーチンについては、サポートされなくなった命令を使用しないように変更する必要があります。この場合は再コンパイルでは効果がないからです。

最初のステップは、命令のエミュレーションが **emstat** ツールを使用することによって起こっているのかどうかの判別です。

emstat プログラムがインストールされていて、使用可能かどうかを判別するには、次のコマンドを実行します。

```
# lslpp -lI bos.perf.tools
```

emstat コマンドは、間隔時間を秒単位で指定し、またオプションで、間隔の数を指定する **vmstat** コマンドと似た働きをします。 最初の欄の値は、システム・ブート以来の累計カウントであるのに対して、2 番目の欄の値はその間隔中にエミュレートされた命令の数です。 1 秒当たり数千ものオーダーでのエミュレーションは、パフォーマンスに影響を与える可能性があります。

emstat 1 コマンドを実行したときの出力の例を以下に示します。

```
# emstat 1

Emulation   Emulation
SinceBoot       Delta
        0           0
        0           0
        0           0
```

エミュレーションが検出されると、次のステップはどのアプリケーションが命令をエミュレートしているかの判別です。この判別は非常に困難です。 1 つの方法は、一度に 1 つのアプリケーションのみを実行して、 **emstat** プログラムでそれをモニターすることです。 特定のエミュレーションでは、トレース・フッ

クにかかることが時々あります。 これは、ASCII トレース・レポート・ファイル内の「PROGRAM CHECK」という語で見ることができます。 このトレース・イベントと関連のあるプロセス/スレッドが命令をエミュレートしているのは、命令をエミュレートするそれ自体のコードによるか、または命令をエミュレートしている他のモジュールのライブラリーまたはコードを実行しているためのどちらかです。

alstat ツールによる桁合わせ例外の検出

データの桁合わせができていない場合は、ハードウェアで桁合わせ例外が発生します。

AIX コンパイラーは、データ型に合わせた桁合わせを行います。 例えば、型がショート (長さは 2 バイト) のデータについては、コンパイラーが自動的に埋め込み、4 バイトにします。 型キャストおよび桁合わせプラグマの使用のような一般的なプログラミング手法では、アプリケーション・データの桁合わせが正しくなくなることがあります。 POWER® プロセッサー・ベース最適化は、データの桁合わせは正しいものと想定しています。 したがって、桁合わせができていないデータのフェッチでは、本来は 1 回のアクセスで済むはずが複数回のメモリー・アクセスが必要となる場合があります。 データの桁合わせができていない場合に生成される桁合わせ例外により、必要なメモリー・アクセスをカーネルがシミュレートせざるを得なくなります。 命令エミュレーションの場合にそうであるように、これはアプリケーションのパフォーマンスを低下させます。

bos.perf.tools にパッケージされた **alstat** ツールを使用して、桁合わせ例外が起きているかどうかを検出することができます。 CPU ごとの桁合わせ例外を表示するには、**-v** オプションを使用します。

alstat と **emstat** は同じバイナリーであるため、これらのツールのいずれかを使用して、命令エミュレーションおよび桁合わせ例外を表示することができます。 命令エミュレーションを表示するには、**-e** オプションを **alstat** で使用します。 桁合わせ例外を表示するには、**-a** オプションを **emstat** で使用します。

alstat の出力は以下のようなものになります。

```
# alstat -e 1
  Alignment  Alignment  Emulation  Emulation
  SinceBoot      Delta  SinceBoot      Delta
          0          0          0          0
          0          0          0          0
          0          0          0          0
```

fdpr プログラムによる実行可能プログラムの再構造化

fdpr (feedback-directed program restructuring) プログラムは、実行可能モジュールを、実行速度を高め、実メモリーを有効に使用できるように最適化します。

fdpr プログラムがインストールされていて、使用可能かどうかを判別するには、次のコマンドを実行します。

```
# lslpp -lI perfagent.tools
```

fdpr コマンドは、ユーザー・レベルのアプリケーション・プログラムのパフォーマンスと実メモリー使用率の両方を改善できる、パフォーマンス・チューニング・ユーティリティーです。 ソース・コードは必ずしも **fdpr** プログラムの入力の形をとるとは限りません。ただし、ストリップされた実行可能プログラムはサポートされません。 ソース・コードが使用可能な場合、**-qfdpr** コンパイラー・フラグ付きで作成されたプログラムには、再配列され、かつ機能が保証されたプログラムの作成の際に **fdpr** プログラムを援助するための情報が含まれます。 **-qfdpr** フラグを使用する場合、プログラム内の全オブジェクト・モジュールに使用する必要があります。 **-qfdpr** フラグを使用した場合、静的リンクではパフォーマンスは改善されません。

fdpr ツールは、次の方法によって、実行可能プログラムの命令を再配列して、命令キャッシュ、変換索引バッファー (TLB)、および実メモリー使用率を改善します。

- 実行頻度が高いコード・シーケンスを 1 つにまとめる (プロファイル作成による判別にしたがって)。
- 条件付き分岐をコーディングし直して、ハードウェア分岐予測を改良する。
- あまり実行されないコードをほかの場所へ移動する。

例えば、「if-then-else」ステートメントがあったとすると、**fdpr** プログラムは、そのプログラムが if 分岐より else 分岐を多く使用すると結論づけることがあります。 すると、条件と 2 つの分岐を次の図のように反転させます。

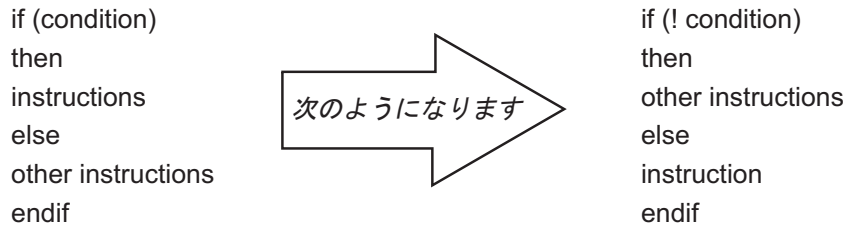


図 *16.* 条件付き分岐の記録の例： この図は、条件付き分岐の再コーディングによって、一部のコードがどのように変更されるかを示しています。 例えば、コード If (condition) は If (! condition) になり、コード then は then のまま、instructions は other instructions になり、else は else のまま、other instructions は instruction になり、endif は endif のままです。

大規模な (5 MB を超える) CPU 制約のアプリケーションでは、実行時間を最大 23% 改善することができますが、一般的には、パフォーマンスの向上は 5% から 20% の間です。 このタイプのプログラムのテキスト・ページの実メモリー要件の低減は、70% に達する場合もあります。 平均は 20% から 50% の間です。 この数字は、アプリケーションの動作と、**fdpr** プログラムを使用したときに指定された最適化オプションによって決まります。

fdpr 処理は、次の 3 つのステージで行われます。

1. 最適化する実行可能モジュールは、詳しいパフォーマンス・データを収集できるように、組み込まれます。
2. 組み込まれた実行可能モジュールは、ユーザーの提供するワークロードの下で実行され、その実行結果のパフォーマンス・データが記録されます。
3. このパフォーマンス・データを使用して、パフォーマンス最適化プロセスが駆動され、その結果再構成された実行可能モジュールは、組み込まれた実行可能プログラムを働かせたワークロードをより効率的に実行できるはずです。 **fdpr** プログラムの駆動に使用されたワークロードは、実際のプログラムの使用状況とよく一致していることが非常に重要です。 再構築された実行可能プログラムに、**fdpr** プログラムの駆動に使用したワークロードと大きく異なるワークロードを使用した場合のパフォーマンスは予想できませんが、元の実行可能プログラムより悪くなる場合があります。

一例として、**# fdpr -p ProgramName -R3 -x test.sh** コマンドは、テスト・ケース `test.sh` を使用して、組み込まれた形式のプログラム **ProgramName** を実行します。その実行の出力は、プログラムの最も積極的な最適化 (**-R3**) の実行に使用され、デフォルトで `ProgramName.fdpr` と呼ばれる、新しいモジュールを形成します。最適化された実行可能プログラムが、最適される前のプログラムより運用においてどの程度優れているかは、テスト・ケース `test.sh` に実際のワークロードがどれだけ反映されているかということに大きく左右されます。

注**: fdpr** プログラムには、高度の最適化アルゴリズムが取り込まれていますが、これによって、元の実行可能モジュールと機能が異なる最適化実行可能プログラムができてしまうことがあります。 どの最適化実行可能プログラムも、実際の運用状態で使用する前に、すなわちその出力を信頼する前に、完全にテストを済ましておくことが絶対に必要です。

要約すると、**fdpr** プログラムのユーザーは、以下のことを厳守する必要があります。

- できるだけ意図した使用法に近いワークロードを使用して、 **fdpr** プログラムを駆動する。
- 最適化された再構成実行可能プログラムの機能を完全にテストする。
- 再構成実行可能プログラムは、チューニングの対象となったワークロードにのみ使用する。

マイクロプロセッサーのコンテンションの制御

AIX カーネルは、スレッドをさまざまなプロセッサーにディスパッチしますが、ほとんどのシステム管理ツールでは、依然としてスレッド自体でなく、スレッドが実行しているプロセスを参照しています。

ユーザー・プロセスの優先順位の制御

ユーザー・プロセスの優先順位は、**nice** または **renice** コマンド、あるいは **setpri()** サブルーチンを使用して操作でき、**ps** コマンドを使用して表示できます。

優先順位の概要は、 44 ページの『プロセスとスレッドの優先順位』に示されています。

優先順位の計算を行う目的は、次のとおりです。

- スレッド間で CPU を共有する。
- スレッドのリソース不足を防ぐ。
- 計算時間を独占するスレッドにペナルティーを適用する。
- 時間の経過に伴ってスレッドを連続的に識別する能力を改善する。

nice コマンドによるコマンドの実行**:**

ユーザーが通常より低い優先順位でコマンドを実行する場合は、**nice** コマンドを使用することができます。

nice コマンドを使用して、通常より高い優先順位でコマンドを実行できるのは、root ユーザーだけです。この場合は、**nice** コマンドの値の範囲は -20 と 19 の間です。

nice コマンドに対して、ユーザーは nice の標準値に加算または減算する値を指定します。変更された nice の値は、指定されたコマンドを実行するプロセスに使用されます。 プロセスの優先順位はまだ固定ではありません。つまり、優先順位値は、CPU 使用率、nice の値、および最小ユーザー・プロセス優先順位値に基づいて、定期的に再計算されます。

フォアグラウンド・プロセスの標準の nice 値は 20 (ksh バックグラウンド・プロセスの場合は 24) です。 次のコマンドを実行すると、**vmstat** コマンドは nice の値を 25 (標準の 20 の代わりに) に指定してフォアグラウンドで実行されます。したがって、優先順位は低くなります。

```
# nice -n 5 vmstat 10 3 > vmstat.out
```

root ログインを使用している場合は、次のコマンドを使用すると、**vmstat** コマンドを高い優先順位で実行できます。

```
# nice -n -5 vmstat 10 3 > vmstat.out
```

root ログインを使用していなかった場合は、前の例の **nice** コマンドを実行すると、**vmstat** コマンドを実行することはできますが、 nice の標準値 20 で実行され、**nice** コマンドはエラー・メッセージをまったく出しません。

setpri サブルーチンによる固定優先順位の設定:

root ユーザー ID の下で実行するアプリケーションは、**setpri()** サブルーチンを使用して、独自の優先順位または別のプロセスの優先順位を設定することができます。

次に例を示します。

```
retcode = setpri(0,59);
```

では、現在のプロセスに固定優先順位 59 を指定します。 **setpri()** サブルーチンで障害が起こった場合は、-1 が戻されます。

次のプログラムでは、優先順位の値とプロセス ID のリストを受け入れ、すべてのプロセスの優先順位を指定値に設定します。

```
/*
   fixprocpri.c
   Usage: fixprocpri priority PID . . .
*/

#include <sys/sched.h>
#include <stdio.h>
#include <sys/errno.h>

main(int argc,char **argv)
{
   pid_t ProcessID;
   int Priority,ReturnP;

   if( argc < 3 ) {
      printf(" usage - setpri priority pid(s) ¥n");
      exit(1);
   }

   argv++;
   Priority=atoi(*argv++);
   if ( Priority < 50 ) {
      printf(" Priority must be >= 50 ¥n");
      exit(1);
   }

   while (*argv) {
      ProcessID=atoi(*argv++);
      ReturnP = setpri(ProcessID, Priority);
      if ( ReturnP > 0 )
          printf("pid=%d new pri=%d  old pri=%d¥n",
            (int)ProcessID,Priority,ReturnP);
      else {
          perror(" setpri failed ");
            exit(1);
      }
   }
}
```

ps コマンドによるプロセス優先順位の表示

ps コマンドの **-l** (小文字の L) フラグは、指定されたプロセスの nice の値と現在の優先順位の値を表示します。

例えば、指定されたユーザーが所有する全プロセスの優先順位を表示するときは、次のように指定します。

```
# ps -lu user1
      F S UID  PID PPID   C PRI NI ADDR    SZ    WCHAN    TTY  TIME CMD
 241801 S 200 7032 7286   0  60 20 1b4c   108           pts/2  0:00 ksh
 200801 S 200 7568 7032   0  70 25 2310    88  5910a58  pts/2  0:00 vmstat
 241801 S 200 8544 6494   0  60 20 154b   108           pts/0  0:00 ksh
```

この出力は、前に説明した **nice -n 5** コマンドの結果を示しています。 プロセス 7568 の優先順位は下位の 70 です。 (**ps** コマンドは、スーパーユーザー・モードで別々のセッションで実行され、したがって 2 つの TTY が存在します。)

プロセスの 1 つが、**setpri(10758, 59)** サブルーチンを使用してそれ自体に固定優先順位を与えた場合、サンプルの **ps -l** の出力は、次のようになります。

```
      F S UID   PID  PPID   C PRI NI ADDR    SZ    WCHAN    TTY  TIME CMD
 200903 S   0 10758 10500   0  59 -- 3438    40  4f91f98  pts/0  0:00 fixpri
```

renice コマンドによる優先順位の変更

renice コマンドは、nice の値を変更し、それに従って、1 つ以上の既に実行中のプロセスの優先順位が変わります。プロセスはプロセス ID、プロセス・グループ ID、またはプロセスを所有するユーザーの名前のいずれかによって識別されます。

renice コマンドは、固定優先順位のプロセスでは使用できません。 root 以外のユーザーは、1 つ以上の実行中プロセスの nice の値に加算する値を指定できますが、減算する値は指定できません。 変更はプロセスの nice の値に対して行われます。これらのプロセスの優先順位は、依然として固定ではありません。root ユーザーだけが、**renice** コマンドを使用して、 -20 から 20 までの範囲の優先度の値を変更するか、または 1 つ以上の実行中プロセスの nice の値から減算することができます。

この例をさらに先に進めるために、**renice** コマンドを使用して、**nice** によって始動した **vmstat** プロセスの nice の値を変更します。

```
# renice -n -5 7568
# ps -lu user1
      F S UID  PID PPID   C PRI NI ADDR    SZ    WCHAN    TTY  TIME CMD
 241801 S 200 7032 7286   0  60 20 1b4c   108           pts/2  0:00 ksh
 200801 S 200 7568 7032   0  60 20 2310    92  5910a58  pts/2  0:00 vmstat
 241801 S 200 8544 6494   0  60 20 154b   108           pts/0  0:00 ksh
```

この結果、プロセスは他のフォアグラウンド・プロセスと等しい、高い優先順位で実行されています。 次のコマンドを出すと、この効果を元に戻すことができます。

```
# renice -n 5 7568
# ps -lu user1
      F S UID  PID PPID   C PRI NI ADDR    SZ    WCHAN    TTY  TIME CMD
 241801 S 200 7032 7286   0  60 20 1b4c   108           pts/2  0:00 ksh
 200801 S 200 7568 7032   1  70 25 2310    92  5910a58  pts/2  0:00 vmstat
 241801 S 200 8544 6494   0  60 20 154b   108           pts/0  0:00 ksh
```

これらの例では、**renice** コマンドは root ユーザーによって実行されました。 通常のユーザー ID で実行する場合は、**renice** コマンドの使用には主に次の 2 つの制限があります。

- そのユーザー ID が所有するプロセスしか指定できない。
- プロセスの nice の値は、**renice** コマンドによって低い優先順位に変更された後は、デフォルトの優先順位に戻す場合でも、減らすことはできない。

nice および renice コマンド構文の説明

nice コマンドと **renice** コマンドでは、nice の標準値 20 に追加する量の指定の方法が異なります。

Nice コマンド	Renice コマンド	得られる nice 値	優先順位の最高値
nice -n 5	**renice -n 5**	25	70
nice -n +5	**renice -n +5**	25	70
nice -n -5	**renice -n -5**	15	55

スレッド優先順位の値の計算

このセクションでは優先順位計算および **schedo** コマンドの使用によるチューニングについて検討します。

136 ページの『schedo コマンド』 コマンドを使用して、各スレッドの優先順位の値に使用される CPU スケジューラー・パラメーターを変更することができます。 優先順位に関する背景情報については、 44 ページの『プロセスとスレッドの優先順位』を参照してください。

schedo プログラムがインストールされていて、使用可能かどうかを判別するには、次のコマンドを実行します。

```
# lslpp -lI bos.perf.tune
```

優先順位の計算**:**

カーネルは、それぞれのスレッドごとに優先順位の値 (スケジューリング優先順位と呼ばれることもある) を維持します。 優先順位の値は正の整数で、関連付けられたスレッドの重要性に逆比例して変化します。つまり、優先順位の値が小さいほど、スレッドの重要性が高いことを意味します。 スケジューラーは、ディスパッチするスレッドを探す際、優先順位の値が最も小さいディスパッチ可能スレッドを選択します。

優先順位の値の計算式は、次のとおりです。

```
priority value = base priority + nice penalty + (CPU penalty based on recent CPU usage)
```

指定されたスレッドの最近の CPU 使用率の値は、タイマー割り込みが発生したときに (10 ミリ秒ごと)、そのスレッドが CPU を管理するたびに、1 だけ加算されます。 最近の CPU 使用率値は、**ps** コマンド出力の *C* 欄に表示されます。 最近の CPU 使用率の最大値は 120 です。

デフォルトのアルゴリズムでは、最近の CPU 使用率を 2 で割ることによって CPU ペナルティーを計算します。したがって、最近の CPU 使用率に対する CPU ペナルティーの比率は 0.5 です。 この比率は、*R* と呼ばれる値 (デフォルトは 16) によって制御されます。 計算式は次のとおりです。

```
CPU_penalty = C * R/32
```

デフォルトのアルゴリズムでは、1 秒に 1 回、すべてのスレッドの最近の CPU 使用率の値を 2 で割ります。したがって、最近の CPU 使用率減衰ファクターは 0.5 です。この比率は、*D* と呼ばれる値 (デフォルトは 16) によって制御されます。 計算式は次のとおりです。

```
C = C * D/32
```

優先順位値の計算アルゴリズムは、プロセス内のスレッドの優先順位を決定するために、プロセスの **nice** 値を使用します。 CPU 時間の単位が増加すると、nice の効果によって優先順位は下がります。 schedo -r -d を使用すると、R と D に新規の値を設定することで優先順位の計算をより細かく制御できます。詳細は、 136 ページの『schedo コマンド』を参照してください。

次の式から始めてください。

```
p_nice = base priority + nice value
```

今度は、次の式を使用します。

```
If p_nice > 60,
   then x_nice = (p_nice * 2) - 60,
   else x_nice = p_nice.
```

nice の値が 20 より大きい場合、優先順位に対する影響は、20 以下の場合の 2 倍になります。新しい優先順位の計算 (整数の切り捨ては無視) は、次のとおりです。

```
priority value = x_nice + [(x_nice + 4)/64 * C*(R/32)]
```

schedo コマンド:

チューニングは、**schedo** コマンドの 2 つのオプション、**sched_R** および **sched_D** によって行われます。

それぞれのオプションは、0 から 32 までの整数をパラメーターとして指定します。これらのパラメーターは、パラメーターの値を掛けてから 32 で割って適用されます。 デフォルトの *R* と *D* の値は 16 で、これは元のアルゴリズム [(D=R=16)/32=0.5] と同じ動作になります。 値の範囲が新しくなった結果、はるかに幅広い動作が可能になりました。 次に例を示します。

```
# schedo -o sched_R=0
```

[(R=0)/32=0、(D=16)/32=0.5] は、CPU ペナルティーが常に 0 であり、優先順位が nice 値だけの関数になることを意味します。 ディスパッチ可能なフォアグラウンド・プロセスが 1 つもない場合を除いて、バックグラウンド・プロセスが CPU 時間を獲得することは絶対にありません。 スレッドの優先順位の値は、技術的には固定優先順位スレッドではありませんが、事実上は一定になります。

```
# schedo -o sched_R=5
```

[(R=5)/32=0.15625, (D=16)/32=0.5] は、フォアグラウンド・プロセスが、**nice -n 10** コマンドで始動されたバックグラウンド・プロセスと、決して競合しないことを意味しています。 累算で 120 CPU 時間の制限は、フォアグラウンド・プロセスの最大ペナルティーが 18 になることを意味します。

```
# schedo -o sched_R=6 -o sched_D=16
```

[(R=6)/32=0.1875, (D=16)/32=0.5] は、バックグラウンド・プロセスが **nice -n 10** コマンドで始動された場合、それは、バックグラウンド・プロセスが任意の CPU 時間を受け取り始める少なくとも 1 秒前になることを意味します。 ただし、フォアグラウンド・プロセスは、CPU 使用率の面でまだ区別できます。本来はバックグラウンドにあるはずのフォアグラウンド・プロセスが長時間実行している場合には、最終的に十分な CPU 使用率が累積され、真のフォアグラウンドに干渉しないようになります。

```
# schedo -o sched_R=32 -o sched_D=32
```

[(R=32)/32=1、(D=32)/32=1] は、長時間実行しているスレッドは C の値が 120 に達するとそこにとどまり、nice の値に基づいて競合することを意味しています。新規スレッドには、十分にタイム・スライスが累積されて、既存スレッドの優先順位の値の範囲に入るまでは、その nice の値に関係なく優先順位を持つことになります。

次に、R と D の値のガイドラインを幾つか示します。

- R の値が小さくなると、優先順位の範囲が狭まり、nice の値が優先順位に与える影響が大きくなる。
- R の値が大きくなると、優先順位の範囲が広がり、nice の値が優先順位に与える影響が小さくなる。
- D の値が小さくなると、CPU 使用率が減衰する速度が速まり、CPU 集中スレッドが早くスケジュールされる可能性がある。
- D の値が大きくなると、CPU 使用率が減衰する速度が遅くなり、CPU 集中スレッドに対するペナルティーが大きくなる (したがって、対話式タイプのスレッドの優先順位が高くなる)。

優先順位計算の例:

この例では、R=4 および D=31 であることが示されており、また実行可能スレッドがほかにないと想定しています。

```
current_effective_priority
                  | base process priority
                  |    | nice value
                  |    |    | count (time slices consumed)
                  |    |    |    | (schedo -o sched_R)
                  |    |    |    |    |
  time 0          p = 40 + 20 + (0   * 4/32) =   60
  time 10 ms      p = 40 + 20 + (1   * 4/32) =   60
  time 20 ms      p = 40 + 20 + (2   * 4/32) =   60
  time 30 ms      p = 40 + 20 + (3   * 4/32) =   60
  time 40 ms      p = 40 + 20 + (4   * 4/32) =   60
  time 50 ms      p = 40 + 20 + (5   * 4/32) =   60
  time 60 ms      p = 40 + 20 + (6   * 4/32) =   60
  time 70 ms      p = 40 + 20 + (7   * 4/32) =   60
  time 80 ms      p = 40 + 20 + (8   * 4/32) =   61
  time 90 ms      p = 40 + 20 + (9   * 4/32) =   61
  time 100ms      p = 40 + 20 + (10  * 4/32) =   61
        .
  (skipping forward to 1000msec or 1 second)
        .
  time 1000ms     p = 40 + 20 + (100 * 4/32) =   72
  time 1000ms     swapper recalculates the accumulated CPU usage counts of
                  all processes. For the above process:
                  new_CPU_usage = 100 * 31/32 = 96  (if d=31)
                  after decaying by the swapper: p = 40 + 20 + ( 96 * 4/32) = 72
                  (if d=16, then p = 40 + 20 + (100/2 * 4/32) = 66)
  time 1010ms     p = 40 + 20 + ( 97 * 4/32) =   72
  time 1020ms     p = 40 + 20 + ( 98 * 4/32) =   72
  time 1030ms     p = 40 + 20 + ( 99 * 4/32) =   72
                 ..
  time 1230ms     p = 40 + 20 + (119 * 4/32) =   74
  time 1240ms     p = 40 + 20 + (120 * 4/32) =   75     count <= 120
  time 1250ms     p = 40 + 20 + (120 * 4/32) =   75
  time 1260ms     p = 40 + 20 + (120 * 4/32) =   75
                 ..
  time 2000ms     p = 40 + 20 + (120 * 4/32) =   75
  time 2000ms     swapper recalculates the counts of all processes.
                  For above process 120 * 31/32 = 116
  time 2010ms     p = 40 + 20 + (117 * 4/32) =   74
```

schedo コマンドによるスケジューラーのタイム・スライスの変更

スケジューラーのタイム・スライスの長さは、**schedo** コマンドで変更可能です。タイム・スライスを変更するには、**schedo -o timeslice**=*value* オプションを使用します。

-t の値はタイム・スライスのティックの数で、SCHED_RR だけがデフォルト以外のタイム・スライス値を使用します (固定優先順位スレッドの説明は、 46 ページの『スレッドのスケジューリング・ポリシー』を参照してください)。

タイム・スライスを変更すると、すぐに有効になり、リブートは必要ありません。

SCHED_OTHER または SCHED_RR のスケジューリング・ポリシーで実行されているスレッドは、クロック・タイムが 10 ms であるとして、タイム・スライス (デフォルトのタイム・スライスは 1 クロック・タイム) の終わりまで CPU を使用することができます。

ある状況においては、コンテキストの切り替えが多すぎて、スレッドのディスパッチによるオーバーヘッドの方が、このようなスレッドを長いタイム・スライスの間実行するよりコストがかかる場合があります。

その場合は、タイム・スライスを長くすれば、固定優先順位スレッドのパフォーマンスによい影響を与える可能性があります。 1 秒当たりのコンテキストの切り替え回数を判別するには、**vmstat** コマンドと **sar** コマンドを使用します。

タイム・スライスを長くした環境では、フル・タイム・スライスを必要としないか、または使用するべきでないアプリケーションもあります。 このようなアプリケーションは、**yield()** システム・コールを使用して、明示的にプロセッサーを解放できます (変更されていない環境でのプログラムと同様)。 **yield()** コールの後、呼び出し側のスレッドは、その優先順位レベルのディスパッチ・キューの最後に移動されます。

mkpasswd コマンドによるマイクロプロセッサー効率のよいユーザー ID 管理

多数のユーザーがいるシステムでログイン応答時間を改善し、マイクロプロセッサー時間を節約するために、オペレーティング・システムでは /etc/passwd ファイルの索引バージョンを使用して、ユーザー ID を検索することができます。 この機能を使用した場合、/etc/passwd ファイルは以前のように存在していますが、通常の処理では使用されません。

索引バージョンのファイルは、**mkpasswd** コマンドによって作成されます。索引バージョンが最新でない場合、ログイン処理は /etc/passwd による、遅いマイクロプロセッサー集中順次検索に戻ります。

mkpasswd -f コマンドを使用して索引パスワード・ファイルを作成します。このコマンドは、/etc/passwd、 /etc/security/passwd、および /etc/security/lastlog の索引付きバージョンを作成します。 作成されるファイルは、/etc/passwd.nm.idx、/etc/passwd.id.idx、/etc/security/passwd.idx、および /etc/security/lastlog.idx です。これにより、暗号化されたパスワードも必要とするアプリケーション (ログインやパスワード認証を必要とする他の任意のプログラム) のパフォーマンスが大きく改善されます。

さらにアプリケーションでは、**getpwent()** の代わりに **_getpwent()**、**getpwnam(name)** の代わりに **_getpwnam_shadow(name,0)**、または **getpwuid(uid)** の代わりに **_getpwuid_shadow(uid,0)** などの代替ルーチンを使用するようにして、暗号化されたパスワードが不要な場合に、名前 /ID の解決を行うことができます。 これによって、/etc/security/passwd のルックアップは行われなくなります。

手作業でパスワード・ファイルを編集しないでください。データベース・ファイル (.idx) のタイム・スタンプの同期が取れなくなり、デフォルトのルックアップ方式 (リニア) が使用されます。 **passwd**、**mkuser**、**chuser**、**rmuser** の各コマンド (または同じ名前の高速パスがある、SMIT コマンドでこれらに相当するもの) がユーザー ID の管理に使用された場合、索引ファイルは自動的に最新に保たれます。 /etc/passwd ファイルがエディターまたは **pwdadm** コマンドで変更されると、索引ファイルの再作成が必要になります。

注**: mkpasswd** コマンドは、NIS、DCE、または LDAP ユーザー・データベースには影響を与えません。

メモリー・パフォーマンス

このセクションでは、メモリー使用の測定と変更の方法について説明します。

システムのメモリーは、ほとんどいつも容量いっぱいまで埋められています。 現在実行中のプログラムが使用可能メモリーをすべて消費していない場合でも、オペレーティング・システムが、前に実行していたプログラムのテキスト・ページや、そこで使用されていたファイルをメモリーに保存しています。 メモリーはいずれにせよ未使用だったので、この保存に関するコストはありません。 多くの場合、プログラムまたはファイルは再び使用されるので、ディスク入出力を削減することができます。

オペレーティング・システムの仮想記憶管理に慣れていない読者は、先に進む前に 50 ページの『仮想メモリー・マネージャーのパフォーマンス』をお読みください。

メモリー使用量

メモリー使用量のレポートを提供するパフォーマンス・ツールが幾つかあります。

最も関心のあるレポートは、**vmstat**、**ps**、および **svmon** コマンドによるものです。

vmstat コマンドによるメモリー使用量の判別

vmstat コマンドは、システム内の全プロセスによって使用されるアクティブ な仮想メモリーの合計の要約ばかりでなく、フリー・リスト上の実メモリー・ページ・フレームの数も示します。

アクティブ仮想メモリーは、実際にタッチされた仮想メモリーの作業セグメント・ページの数で定義されます。 この数は、アクティブ仮想メモリー・ページの一部がページング・スペースに書き出されている可能性があるので、マシン内の実ページ・フレームより大きい場合があります。

システムでメモリーが不足しているかまたは一部のメモリー・チューニングが必要な場合、設定された間隔だけ **vmstat** コマンドを実行して、得られたレポートの「*pi*」欄と「*po*」欄を調べます。 これらの欄は、1 秒当たりのページインするページング・スペースの数、および 1 秒当たりのページアウトするページング・スペースの数を示しています。 この値が常にゼロ以外の場合は、メモリーのボトルネックが存在する恐れがあります。 時々ゼロ以外の値になるのは、ページングが仮想メモリーの基本原理なので問題ありません。

```
# vmstat 2 10
kthr     memory             page              faults        cpu
----- ----------- ------------------------ ------------ -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 1  3 113726   124   0  14   6 151  600   0 521 5533 816 23 13  7 57
 0  3 113643   346   0   2  14 208  690   0 585 2201 866 16  9  2 73
 0  3 113659   135   0   2   2 108  323   0 516 1563 797 25  7  2 66
 0  2 113661   122   0   3   2 120  375   0 527 1622 871 13  7  2 79
 0  3 113662   128   0  10   3 134  432   0 644 1434 948 22  7  4 67
 1  5 113858   238   0  35   1 146  422   0 599 5103 903 40 16  0 44
 0  3 113969   127   0   5  10 153  529   0 565 2006 823 19  8  3 70
 0  3 113983   125   0  33   5 153  424   0 559 2165 921 25  8  4 63
 0  3 113682   121   0  20   9 154  470   0 608 1569 1007 15  8  0 77
 0  4 113701   124   0   3  29 228  635   0 674 1730 1086 18  9  0 73
```

上記の出力例では、出力に入出力待ちが多いことや、ブロックされたキューのスレッドの数に注意してください。 その他の入出力アクティビティーが入出力待ちを発生させることがありますが、今回のケースでは、ほとんどの入出力待ちはページング・スペースからのページインおよびページアウトによるものです。

システムにその VMM に関するパフォーマンス上の問題があるかどうかを見るには、「*memory*」と「*page*」の下の欄を調べます。

- **memory**

  実メモリーおよび仮想メモリーに関する情報を提供します。

  - **avm**

    アクティブ仮想メモリーの avm 欄は、**vmstat** サンプルが収集された時点でアクティブであった仮想メモリー・ページ数を表しています。 据え置きページ・スペース・ポリシーはデフォルト・ポリシーです。 このポリシーの下では、avm 値は、使用されているページング・スペース・ページ数より大きくなることがあります。 avm 統計情報に、ファイル・ページは含まれません。

  - **fre**

「fre」欄は、空きメモリー・ページの平均数を示します。 1 ページは、実メモリーの 4 KB の領域です。 システムは、メモリー・ページのバッファーを維持しています。これはフリー・リストと呼ばれ、VMM にスペースが必要な場合はすぐにアクセスできるようになっています。 VMM がフリー・リストに保持する最小ページ数は、**vmo** コマンドの *minfree* パラメーターによって決まります。 詳細については、 164 ページの『VMM ページ置換のチューニング』を参照してください。

アプリケーションが終了すると、その作業ページはすべて、即時にフリー・リストに戻されます。ただし、その永続ページまたはファイルは RAM に残されるので、VMM が他のプログラムのためにスチールするまでは、フリー・リストに戻されません。 永続ページは、対応するファイルが削除された場合にも解放されます。

このため、「*fre*」値は、プロセスがいつでも使用できる実メモリーをすべて示しているとは限りません。 ページ・フレームが必要な場合、終了したアプリケーションに関連する永続ページは、最初に別のプログラムに渡されるものの中の 1 つです。

「*fre*」値が *maxfree* 値をかなり上回っている場合は、システムがスラッシング 状態であるということはありません。 スラッシングとは、システムが連続してページインとページアウトを行っていることを意味します。 ただし、システムがスラッシング状態になっている場合、「*fre*」値が小さくなることは確かです。

- **page**

  ページ・フォールトとページング・アクティビティーに関する情報。 これらは間隔内で平均され、1 秒当たりの単位で示されます。

  - **re**

    注**:** この欄は現在サポートされません。

  - **pi**

    pi 欄には、ページング・スペースからページインされるページの数が詳しく表示されます。 ページング・スペースは、仮想メモリーの一部で、ディスクに常駐している部分です。 これは、メモリーがオーバーコミットされたときのオーバーフローとして使用されます。 ページング・スペースは、実メモリーからスチールされた作業セット・ページのストレージ専用の論理ボリュームで構成されています。 プロセスがスチールされたページを参照すると、ページ・フォールトが起こるので、そのページをページング・スペースからメモリーに読み込む必要があります。

    ハードウェア、ソフトウェアおよびアプリケーションの構成が多様なので、注意すべき絶対的数は存在しません。 このフィールドは、ページング・スペース・アクティビティーのキー・インディケーターとして重要です。 ページインが発生した場合は、その前にそのページのページアウトがあったはずです。 メモリー制約環境では、各ページインによって強制的に異なるページがスチールされ、その結果、ページアウトが行われることもよくあります。

  - **po**

    「*po*」欄には、ページング・スペースにページアウトされるページの数 (率) が表示されます。 作業用ストレージのページがスチールされると必ず、それがまだページング・スペースに常駐していないかまたは変更されている場合は、ページング・スペースに書き込まれます。そして再度参照されなければ、プロセスが終了するかスペースを放棄するまで、ページング・デバイスに存在し続けます。その後で不在になったページに含まれているアドレスを参照すると、ページ・フォールトになり、そのページはシステムによって個々にページインされます。 プロセスが正常に終了すると、そのプロセスに割り当てられたページング・スペースは解放されます。 システムが多数の永続ページを読み

込んでいる場合、対応する *pi* が増加しないのに、*po* が増加することに気付く場合があります。 これは必ずしもスラッシングを示しているわけではありませんが、アプリケーションのデータ・アクセスのパターンを調査する必要があります。

– **fr**

その間隔中に、ページ置換アルゴリズムによって解放された 1 秒当たりのページ数。 VMM ページ置換ルーチンは、ページ・フレーム・テーブル (つまり PFT) をスキャンするとき、使用可能メモリー・フレームのフリー・リストの補充に、どのページをスチールするかの選択の基準を使用します。この基準には、両方の種類のページ、作業 (計算) およびファイル (永続) ページが含まれます。 ページが解放されているだけでは、何らかの入出力が行われたことを意味することにはなりません。例えば、永続ストレージ (ファイル) ページが変更されていない場合、そのページは元のディスクに書き出されません。 入出力が不要であれば、ページの解放に最低のシステム・リソースしか必要としません。

– **sr**

その間隔中に、ページ置換アルゴリズムによって調べられた 1 秒当たりのページ数。 ページ置換アルゴリズムは、ページ置換しきい値の要件を満たすほど十分にスチールするために、多くのページ・フレームをスキャンしなければならない場合があります。 **sr** 値が **fr** 値よりも高くなればなるほど、ページ置換アルゴリズムがスチール対象の適格なページを見つけるのが難しくなります。

– **cy**

クロック・アルゴリズムのサイクル数/秒。 VMM は、クロック・アルゴリズムと呼ばれている手法を使用して、置き換えるページを選択します。この手法は、各ページの被参照ビットを利用して、最近どのページが使用 (参照) されたかを示すものです。 ページ・スチール・ルーチンは、呼び出されると、PFT 内を巡回して、各ページの被参照ビットを調べます。

「*cy*」欄は、ページ置換コードが PFT をスキャンした回数/秒を示します。 フリー・リストは PFT のスキャンを完了せずに補充することができ、また **vmstat** フィールドはすべて整数で報告されるので、このフィールドは通常はゼロになっています。

システムに適した RAM の量を判別する 1 つの方法は、**vmstat** コマンドによって報告された *avm* の最大値を調べることです。それに 4 K をかけてバイト数を求めてから、結果をシステム上の RAM のバイト数と比較します。 理想的には、*avm* は合計 RAM より少なくする必要があります。 多い場合は、多少の仮想メモリーのページングが行われます。 発生するページングの量は、2 つの値の差によって決まります。 仮想メモリーのアイデアは、実際にある以上のメモリーのアドレッシングの能力が得られる (メモリーの一部は RAM にあり、残りはページング・スペースにある) ということです。 しかし、実メモリーをはるかに超える仮想メモリーが存在する場合は、この方法によって大量のページングが行われ、その結果遅延が発生する恐れがあります。 *avm* が RAM より少ない場合は、RAM がファイル・ページでいっぱいになったために、ページング・スペースのページングが起こる可能性があります。 そのような場合は、*minperm*、*maxperm*、 *maxclient* の値をチューニングすると、ページング・スペースのページングの量を減らすことができます。 詳しくは、 164 ページの『VMM ページ置換のチューニング』を参照してください。

vmstat -I コマンド**:**

vmstat -I コマンドは、ファイル・ページイン/秒、ファイル・ページアウト/秒 (ページング・スペースのページインまたはページアウトでない、任意の VMM ページインまたはページアウト) などの、追加情報を表示します。

「re」欄と「cy」欄は、このフラグでは報告されません。

vmstat -s コマンド:

要約オプション **(-s)** は、要約レポートを標準出力に送信するオプションで、システム初期化時に開始し、間隔ベースでなく絶対カウントで表します。

推奨されるこれらの統計情報の使用方法は、ワークロードの前にこのコマンドを実行し、出力を保管してから、ワークロードの後に再び実行して、その出力を保管します。 次のステップでは、2 組の出力の間の相違を判別します。 これを自動的に行う、**vmstatit** と呼ばれる **awk** スクリプトは、 39 ページの『ディスクまたはメモリー関連の問題』に記載されています。

```
# vmstat -s
  3231543 total address trans. faults
    63623 page ins
   383540 page outs
      149 paging space page ins
      832 paging space page outs
        0 total reclaims
   807729 zero filled pages faults
     4450 executable filled pages faults
   429258 pages examined by clock
        8 revolutions of the clock hand
   175846 pages freed by the clock
    18975 backtracks
        0 lock misses
       40 free frame waits
        0 extend XPT waits
    16984 pending I/O waits
   186443 start I/Os
   186443 iodones
141695229 cpu context switches
317690215 device interrupts
        0 software interrupts
        0 traps
 55102397 syscalls
```

要約の中のページインとページアウトの数は、ページ・スペースとファイル・スペースに関する、仮想メモリーのページインとページアウトのアクティビティーを表しています。 ページング・スペースのページインとページアウトは、ページ・スペースのみを表しています。

ps コマンドによるメモリー使用量の判別

ps コマンドは、個々のプロセスのメモリー使用率のモニターにも使用できます。

ps v *PID* コマンドは、次のように、個々のプロセスのメモリー関連の統計情報の最も広範囲の報告を提供します。

- ページ・フォールト
- タッチされた作業セグメントのサイズ
- メモリー内の作業セグメントとコード・セグメントのサイズ
- テキスト・セグメントのサイズ
- 常駐セットのサイズ
- このプロセスで使用される実メモリーの割合

次に例を示します。

```
# ps v
   PID    TTY STAT  TIME PGIN  SIZE   RSS   LIM  TSIZ   TRS %CPU %MEM COMMAND
 36626  pts/3 A     0:00    0   316   408 32768    51    60  0.0  0.0 ps v
```

得られた **ps** レポートの最も重要な欄について、以下に説明します。

PGIN ページ・フォールトが原因で起こったページインの数。 入出力はすべてページ・フォールトと分類されているので、これは基本的には入出力ボリュームの測定値です。

SIZE プロセスのデータ・セクションの K バイト単位の仮想サイズ (ページング・スペース内) (他のフラグの場合は SZ と表示される)。 この数は、タッチされたプロセスの作業セグメント・ページ数に 4 をかけた数と同じです。一部の作業セグメント・ページが現在ページアウトされている場合、この数は使用している実メモリーの量より大きくなります。 SIZE には、専用セグメントとプロセスの共用ライブラリー・データ・セグメントのページが含まれます。

RSS プロセスの実メモリー (常駐セット) サイズ (K バイト)。 この数は、メモリー内の作業セグメントとコード・セグメントの数の合計を 4 倍した値と同じです。コード・セグメント・ページは、プログラムの現在実行中の全インスタンスで共用されていることを覚えておいてください。 26 の **ksh** プロセスが実行中の場合、**ksh** 実行可能プログラムの任意のページの 1 つのコピーだけがメモリーにありますが、**ps** コマンドは、そのコード・セグメント・サイズを **ksh** プログラムの各インスタンスの RSS の一部として報告します。

TSIZ テキスト (共用プログラム) イメージのサイズ。 これは、実行可能ファイルのテキスト・セクションのサイズです。 実行可能プログラムのテキスト・セクションのページは、タッチされているときだけ、すなわち、そこに分岐しているか、そこから読み込んでいるときだけメモリーに入れられます。 この数は、読み込むことができるテキストの量の上限を表しているだけです。 TSIZ 値は、実際に使用されているメモリーを反映していません。この TSIZ 値は、**dump -ov** コマンドを実行可能プログラムに対して実行しても、表示することができます (例えば、**dump -ov** `/usr/bin/ls`)。

TRS テキストの常駐セット (実メモリー) のサイズ。 これは、コード・セグメント・ページの数の 4 倍です。この数は、複数インスタンスが実行しているプログラムのメモリー使用を過大に表します。 TRS 値は、XCOFF ヘッダーやローダー・セクションなどのコード・セグメントに他のページが含まれている場合があるので、TSIZ 値より高くなる可能性があります。

%MEM
メモリー内の作業セグメントとコード・セグメントのページ数の和を 4 倍して (RSS 値)、マシン内の使用中の実メモリーのサイズ (KB 単位) で割り、100 をかけ、至近の整数のポイントに丸めます。 この値は、プロセスが使用しようとする実メモリーのパーセンテージを伝えようとします。 運悪く、RSS と同様、他のプロセスとプログラム・テキストを共用しているプロセスのコストを、大きく見積もりすぎる傾向があります。 さらに、至近のパーセンテージに切り上げると、システム内の 0.005 未満の RSS 値を持つシステムのプロセスは、すべて、実メモリーのサイズをかけると %MEM が 0.0 になります。

注**: ps** コマンドは、共有メモリー・セグメントまたはメモリー・マップ・セグメントが消費するメモリーは示しません。 多くのアプリケーションは共有メモリーまたはメモリー・マップのセグメントを使用するので、これらのセグメントのメモリー使用を表示するには、**svmon** コマンドの方が適したツールです。

svmon コマンド

svmon コマンドを使用すると、メモリー使用率のより深い分析が可能です。 このコマンドの方が **vmstat** および **ps** コマンドより情報が豊富ですが、割り込みが多くなります。 **svmon** コマンドは、現在のメモリーの状態のスナップショットをキャプチャーします。 ただし、割り込みが使用可能なユーザー・レベルで実行されるので、本当のスナップショットではありません。

svmon がインストールされていて、使用可能かどうかを判別するには、次のコマンドを実行します。

```
# lslpp -lI bos.perf.tools
```

svmon コマンドは、root ユーザーのみが実行可能です。

-i オプションで間隔が使用されている場合、統計情報は、コマンドが削除されるまでか、間隔の数 (これは間隔のすぐ後に指定できる) に達するまで表示されます。

以下のさまざまなレポートを使用して、表示されている情報を分析することができます。

Global (-G)
システム全体で使用中の実メモリーおよびページング・スペースを記述した統計情報を表示します。

Process (-P)
指定されたアクティブ・プロセスのメモリー使用率情報を表示します。 プロセスのリストが指定されていない場合、メモリー使用率統計情報には、すべてのアクティブ・プロセスが表示されます。

Segment (-S)
指定されたセグメントのメモリー使用率情報を表示します。 セグメントのリストが指定されていない場合、メモリー使用率統計情報には、定義されているすべてのセグメントが表示されます。

Detailed Segment (-D)
指定されたセグメントの詳細情報を表示します。

User (-U)
指定されたログイン名のメモリー使用率統計情報を表示します。 ログイン名のリストが指定されていない場合、メモリー使用率統計情報には、定義されているすべてのログイン名が表示されます。

Command (-C)
コマンド名によって指定されたプロセスのメモリー使用率統計情報を表示します。

Workload Management Class (-W)
指定されたワークロード・マネージメント・クラスのメモリー使用率統計情報を表示します。 クラスが提供されていない場合、メモリー使用率統計情報には、定義されているすべてのクラスが表示されます。

Frame (-F)
フレームに関する情報を表示します。 フレーム番号が指定されていない場合、使用メモリーのパーセンテージが報告されます。 参照ビットが設定されるのは、考慮されるフレームだけです。 処理期間中に、すべての参照ビットがリセットされます。 したがって、**-f** オプションを 2 度目に使用すると、**svmon** コマンドは、前回 **-f** オプションが使用された後でアクセスされた実メモリーのパーセンテージを報告します。 システムに予約済みプールが定義されている場合は、定義済みのプールで使用されているメモリーのパーセンテージが報告されます。

Tier (-T)
tier 番号、**-a** フラグが使用された場合のスーパークラス名、および tier に属するセグメントからの実メモリーの合計ページ数などの、tier に関する情報を表示します。

使用中のメモリー量**:**

svmon コマンドは使用中のメモリー量に関するデータを提供します。

グローバル統計情報を表示するには、**-G** フラグを使用します。 次の例では、1 秒間隔で 2 回繰り返されます。

```
# svmon -G -i 1 2

               size      inuse       free        pin    virtual
memory      1048576     425275     623301      66521     159191
pg space     262144      31995

               work       pers       clnt
pin           46041          0          0
in use       129600     275195          0

PageSize   PoolSize      inuse       pgsp        pin    virtual
s   4 KB          -     404795      31995      46041     159191
L  16 MB          5          0          0          5          0
               size      inuse       free        pin    virtual
memory      1048576     425279     623297      66521     159195
pg space     262144      31995

               work       pers       clnt
pin           46041          0          0
in use       129604     275195          0

PageSize   PoolSize      inuse       pgsp        pin    virtual
s   4 KB          -     404799      31995      46041     159195
L  16 MB          5          0          0          5          0
```

システム上で使用できるページが 4 KB しかない場合、ページ・サイズごとに情報を分断するセクションは表示されません。

得られた **svmon** レポートの欄について、以下に説明します。

memory
実メモリーの使用を記述する統計情報を 4 KB ページ単位で表示します。

size メモリーの合計サイズ (4 KB ページ単位)。

inuse RAM 内の、プロセスが使用しているページ数と終了したプロセスに属していて、まだ RAM にある永続ページの数を加えた数。 この値は、メモリーの合計サイズからフリー・リスト上のページの数を引いた数です。

free フリー・リスト上のページの数。

pin RAM に固定されているページの数 (固定されたページとは、常に RAM に常駐していて、ページアウトできないページのことです)。

virtual
プロセス仮想スペース内に割り当てられたページの数。

pg space
ページング・スペースの使用を記述する統計情報を 4 KB ページ単位で表示します。 報告された値は、実際に使用されたページング・スペースの数です。これは、これらのページがページング・スペースにページアウトされたことを示します。 これは **vmstat** コマンドとは異なります。 **vmstat** コマンドの avm 欄には、アクセスされるが、必ずしもページアウトはされない仮想メモリーが表示されます。

size ページング・スペースの合計サイズ (4 KB ページ単位)。

inuse 割り当てられたページの合計数。

pin 固定ページを含む実メモリーのサブセットに関する詳細統計情報を 4 KB フレーム単位で表示します。

work RAM 内の固定された作業ページの数。

pers RAM 内の固定された永続ページの数。

clnt RAM 内の固定されたクライアント・ページ数。

in use

使用中の実メモリーのサブセットに関する詳細統計情報を 4 KB フレーム単位で表示します。

work RAM 内の作業ページの数。

pers RAM 内の永続ページの数。

clnt RAM 内のクライアント・ページ数 (クライアント・ページはリモート・ファイル・ページです)。

PageSize

システム上で 4 KB 以外のページ・サイズが使用可能な場合にのみ表示されます。システム上で使用可能なページ・サイズごとの個々の統計を指定します。

PageSize
ページ・サイズ

PoolSize
予約済みメモリー・プール内のページ数。

inuse 使用中のページ数

pgsp ページング・スペース内に割り当てられているページ数

pin ピンされたページ数

virtual
システム仮想スペース内に割り当てられたページ数。

この例では、1 048 576 ページの合計メモリー・サイズがあります。 この数に 4096 を掛けると、合計実メモリー・サイズ (バイト単位) が分かります (4 GB)。 425 275 ページが使用中であれば、フリー・リストに 623 301 ページあり、66 521 ページが RAM に固定されています。 使用中の合計ページのうち、RAM 内に 129 600 の作業ページ、RAM 内に 275 195 の永続ページがあり、クライアント・ページは RAM 内にありません。 これら 3 つの部分と、予約済みプールが必ずしも使用しない予約されたメモリーの合計は、「*memory*」部分の「*inuse*」欄と同じです。「*pin*」部分は、固定メモリー・サイズを作業、永続、およびクライアントのカテゴリーに分割します。 それらと、予約済みプールによって予約されているメモリー (常にピンされています) の合計は、*memory* 部分の *pin* 欄と同じです。合計ページング・スペースは 262 144 ページ (1 GB) あり、31 995 ページが使用中です。 「*memory*」の「*inuse*」欄は、プログラムの完了時に、ページング・スペース割り当てが解放されているのに、ファイル・ページのメモリーが解放されていないために、通常は「*pg space*」の「*inuse*」欄より大きい値です。

プロセスごとのメモリー使用量**:**

svmon -P コマンドは、現在システムで実行中のすべてのプロセスのメモリー使用率統計情報を表示します。

以下に、**svmon -P** コマンドの例を示します。

```
# svmon -P

--------------------------------------------------------------------------------
     Pid Command          Inuse        Pin     Pgsp    Virtual 64-bit Mthrd 16MB
   16264 IBM.ServiceRM    10075       3345     3064      13310      N     Y    N

     PageSize      Inuse        Pin       Pgsp    Virtual
     s   4 KB      10075       3345       3064      13310
```

```
    L  16 MB          0          0          0          0

    Vsid      Esid Type Description              PSize  Inuse   Pin Pgsp Virtual
   f001e         d work shared library text          s   4857     0   36  6823
       0         0 work kernel seg                  s   4205  3335 2674  5197
   b83f7         2 work process private             s    898     2  242  1098
   503ea         f work shared library data         s     63     0   97   165
   c8439         1 pers code,/dev/hd2:149841        s     28     0    -     -
   883f1         - work                             s     21     8   14    26
   e83dd         - pers /dev/hd2:71733              s      2     0    -     -
   f043e         4 work shared memory segment       s      1     0    1     1
   c0438         - pers large file /dev/hd9var:243  s      0     0    -     -
   b8437         3 mmap mapped to sid a03f4         s      0     0    -     -
   583eb         - pers large file /dev/hd9var:247  s      0     0    -     -

-------------------------------------------------------------------------------
     Pid Command          Inuse        Pin     Pgsp    Virtual 64-bit Mthrd 16MB
   17032 IBM.CSMAgentR     9791       3347     3167      12944      N     Y    N

     PageSize      Inuse        Pin       Pgsp    Virtual
     s   4 KB       9791       3347       3167      12944
     L  16 MB          0          0          0          0

    Vsid      Esid Type Description              PSize  Inuse   Pin Pgsp Virtual
   f001e         d work shared library text          s   4857     0   36  6823
       0         0 work kernel seg                  s   4205  3335 2674  5197
     400         2 work process private             s    479     2  303   674
   38407         f work shared library data         s    120     0  127   211
   a83f5         1 pers code,/dev/hd2:149840        s     99     0    -     -
   7840f         - work                             s     28    10   27    39
   e83dd         - pers /dev/hd2:71733              s      2     0    -     -
   babf7         - pers /dev/hd2:284985             s      1     0    -     -
   383e7         - pers large file /dev/hd9var:186  s      0     0    -     -
   e03fc         - pers large file /dev/hd9var:204  s      0     0    -     -
   f839f         3 mmap mapped to sid 5840b         s      0     0    -     -
[...]
```

このコマンド出力は、プロセスごとのグローバル・メモリー使用量と、報告される各プロセスで使用されるセグメントごとのメモリー使用量の、詳細情報を示します。 デフォルトのソート規則では、Inuse ページ数の多い順にソートされます。 ソート規則は、**svmon** コマンドの **-u**、**-p**、**-g**、または **-v** フラグを使用して変更できます。

システムのメモリーを使用する上位 15 プロセスの要約を表示するには、次のコマンドを使用します。

```
# svmon -Pt15 | perl -e 'while(<>){print if($.==2||$&&&!$s++);$.=0 if(/^-+$/)}'
-------------------------------------------------------------------------------
     Pid Command          Inuse        Pin     Pgsp    Virtual 64-bit Mthrd 16MB
   16264 IBM.ServiceRM    10075       3345     3064      13310      N     Y    N
   17032 IBM.CSMAgentR     9791       3347     3167      12944      N     Y    N
   21980 zsh               9457       3337     2710      12214      N     N    N
   22522 zsh               9456       3337     2710      12213      N     N    N
   13684 getty             9413       3337     2710      12150      N     N    N
   26590 perl5.8.0         9147       3337     2710      12090      N     N    N
    7514 sendmail          9390       3337     2878      12258      N     N    N
   14968 rmcd              9299       3340     3224      12596      N     Y    N
   18940 ksh               9275       3337     2710      12172      N     N    N
   14424 ksh               9270       3337     2710      12169      N     N    N
    4164 errdemon          9248       3337     2916      12255      N     N    N
    3744 cron              9217       3337     2770      12125      N     N    N
   11424 rpc.mountd        9212       3339     2960      12290      N     Y    N
   21564 rlogind           9211       3337     2710      12181      N     N    N
   26704 rlogind           9211       3337     2710      12181      N     N    N
```

Pid 16 264 は、メモリー使用量が最も多いプロセスの ID です。 Command はコマンド名を表しており、この例では IBM.ServiceRM となっています。プロセスによって使用されるセグメントの実メモリー・ペー

ジの合計数である「Inuse」欄は、10 075 ページを示しています。 各ページは 4 KB です。 プロセスが使用するセグメントからページインされるページの合計数である「Pin」欄は、3 345 ページを示しています。 プロセスが使用するページング・スペース・ページの合計数である「Pgsp」欄は、3 064 ページを示しています。 「Virtual」欄 (プロセス仮想スペース内の合計ページ数) は、13 310 を示しています。

詳細セクションには、要約セクションに示されている各プロセスの各セグメントに関する情報が表示されます。 これには、仮想 Vsid および実効 Esid セグメント ID が含まれます。 Esid は、対応するページのアクセスに使用されるセグメント・レジスターを反映しています。 セグメントのタイプも、その記述 (永続セグメントのファイルのボリューム名と i ノードを含む、セグメントのテキスト記述) とともに表示されます。 レポートには、セグメントがサポートされているページのサイズ、RAM 内のページ数 Inuse、RAM 内のピンされたページ数 Pin、ページング・スペース内のページ数 Pgsp、および仮想ページ数 Virtual についても詳しく示されます。ここで、s は 4 KB ページ、L は 16 MB ページを表します。

さらにオプションを使用して、より詳細な情報を表示することができます。 **-j** オプションを使用すると、永続セグメントのファイルのパスが表示されます。 **-l** オプションを使用するとセグメントの詳細情報が、**-r** オプションを使用すると各セグメントで使用されているメモリー範囲が、それぞれ表示されます。 **svmon** コマンドで **-l**、**-r**、および **-j** オプションを指定した場合の例を以下に示します。

```
# svmon -S f001e 400 e83dd -l -r -j

    Vsid      Esid Type Description              PSize  Inuse   Pin Pgsp Virtual
   f001e         d work shared library text          s   4857     0   36  6823
                   Addr Range: 0..60123
                   Shared library text segment
     400         2 work process private             s    480     2  303   675
                   Addr Range: 0..969 : 65305..65535
                   pid(s)=17032
   e83dd         - pers /dev/hd2:71733              s      2     0    -     -
                        /usr/lib/nls/loc/uconvTable/ISO8859-1
                   Addr Range: 0..1
                   pid(s)=17552, 17290, 17032, 16264, 14968, 9620
```

「Address Range」は、永続セグメントまたはクライアント・セグメントの場合は 1 つの範囲を、作業範囲の場合は 2 つの範囲を指定します。 永続またはクライアント・セグメントの場合の範囲は、「0..x,」の形式をとりますが、この x は今まで使用された仮想ページの最大数です。 作業セグメントの範囲フィールドは、「0..x : y..65535」とすることができます。0..x にはグローバル・データが含まれていて、上方に向かって増え、y..65535 にはスタック領域が含まれていて、下方に向かって増えます。 アドレス範囲の場合、作業セグメントでは、スペースは両方の端から割り当てられ、中央に向かって進みます。 作業セグメントが専用でない場合 (カーネルまたは共用ライブラリー)、スペースは別の方法で割り当てられます。

上の例では、セグメント ID 400 は専用作業セグメントで、そのアドレス範囲は 0..969 : 65305..65535 です。 セグメント ID f001e は共用ライブラリー・テキスト作業セグメントで、そのアドレス範囲は 0..60123 です。

セグメントは、複数のプロセスで使用することができます。 このようなセグメントの実メモリー内の各ページは、そのセグメントを使用する各プロセスの「Inuse」フィールドに含まれます。 したがって、「Inuse」の合計が、実メモリー内のページ数の合計を超える可能性があります。 「Pgsp」および「Pin」フィールドについても、同じことが言えます。要約セクションに表示される値は、プロセスによって使用されるすべてのセグメントの Inuse、Pin、および Pgsp についてのそれぞれの合計と、Virtual カウンターから成ります。

この例では、e83dd セグメントは、PID が 17552、17290、17032、16264、14968、および 9620 の、いくつものプロセスによって使用されています。

特定セグメント **ID** の詳細情報**:**

-D オプションは、セグメントの詳細メモリー使用率統計情報を表示します。

次に例を示します。

```
# svmon -D 38287 -b
Segid: 38287
Type:  working
PSize: s (4 KB)
Address Range: 0..484
Size of page space allocation: 2 pages (  0,0 MB)
Virtual: 18 frames ( 0,1 MB)
Inuse: 16 frames ( 0,1 MB)

          Page   Psize    Frame     Pin        Ref        Mod  ExtSegid    ExtPage
           341    s       527720     N          N          N         -          -
           342    s       996079     N          N          N         -          -
           343    s       524936     N          N          N         -          -
           344    s       985024     N          N          N         -          -
           347    s       658735     N          N          N         -          -
           348    s        78158     N          N          N         -          -
           349    s       174728     N          N          N         -          -
           350    s       758694     N          N          N         -          -
           404    s       516554     N          N          N         -          -
           406    s       740622     N          Y          N         -          -
           411    s       528313     N          Y          Y         -          -
           412    s      1005599     N          Y          N         -          -
           416    s       509936     N          N          Y         -          -
           440    s       836295     N          N          Y         -          -
           443    s        60204     N          N          Y         -          -
           446    s       655288     N          N          Y         -          -
```

これらの欄の説明を以下に示します。

Page セグメント内のページの索引を指定します。

Psize ページのサイズを指定します (4 KB は **s**、64 KB は **m**、16 MB は **L**、16 GB は **S**)。

Frame
ページがある実メモリー・フレームの索引を指定します。

Pin ページが固定されているかどうかを示すフラグを指定します。

Ref **-b** フラグでのみ指定されます。 ページの参照ビットがオンかどうかを示すフラグを指定します。

Mod **-b** フラグでのみ指定されます。 ページが変更されているかどうかを示すフラグを指定します。

ExtSegid
ページが検査対象セグメントにリンクされた拡張セグメントに属している場合に、そのセグメントの仮想セグメント ID が表示されます。

ExtPage
ページが検査対象セグメントにリンクされた拡張セグメントに属している場合に、その拡張セグメント内のページの索引が表示されます。

拡張セグメントが検査対象セグメントにリンクされている場合、レポートは、以下の例のようになります。

```
           Page     Psize   Frame     Pin        Ref        Mod  ExtSegid    ExtPage
          65574     s       345324     N          N          N    288071         38
          65575     s       707166     N          N          N    288071         39
          65576     s       617193     N          N          N    288071         40
```

-b フラグは、表示されているすべてのフレームの参照ビットと変更ビットの状況を表示します。 これが示された後で、フレームの参照ビットはリセットされます。 **-i** フラグと共に使用されると、各間隔の間にどのフレームがアクセスされるかを検出します。

注**: -b** フラグはパフォーマンスに影響するので、使用するときは注意してください。

セグメントのメモリー使用率上位のリスト**:**

-S オプションは、メモリー使用率によるセグメントのソートと、指定したセグメントのメモリー使用率統計情報の表示に使用されます。 セグメントのリストが指定されていない場合、メモリー使用率統計情報には、定義されているすべてのセグメントが表示されます。

次のコマンドは、システムおよび非システム・セグメントを実メモリー内のページ数によってソートします。 **-t** オプションを使用することで、表示するセグメント数を指定の数に制限することができます。 **-u** フラグは、出力結果を実メモリー内のページ数の合計の降順でソートします。

svmon コマンドで、**-S**、**-t**、**-u** の各オプションを指定した場合の出力例を以下に示します。

```
# svmon -Sut 10

    Vsid      Esid Type Description              PSize  Inuse   Pin Pgsp Virtual
   70c4e         - pers large file /dev/lv01:26      s  84625     0    -     -
   22ec4         - work                              s  29576     0    0 29586
   8b091         - pers /dev/hd3:123                 s  24403     0    -     -
   7800f         - work kernel heap                  s  22050  3199 19690 22903
   a2db4         - pers /dev/hd3:105                 s  15833     0    -     -
   80010         - work page frame table             s  15120 15120    0 15120
   7000e         - work misc kernel tables           s  13991     0 2388 14104
   dc09b         - pers /dev/hd1:28703               s   9496     0    -     -
   730ee         - pers /dev/hd3:111                 s   8568     0    -     -
   f001e         - work                              s   4857     0   36  6823
```

svmon コマンドと vmstat コマンドの出力の相関

svmon と **vmstat** の出力の間には相関があります。

以下は、**svmon** コマンドの出力例です。

```
# svmon -G
               size      inuse       free        pin    virtual
memory      1048576     417374     631202      66533     151468
pg space     262144      31993

               work       pers       clnt
pin           46053          0          0
in use       121948     274946          0

PageSize   PoolSize      inuse       pgsp        pin    virtual
s    4 KB          -     397194     262144      46053     151468
L   16 MB          5          0          0          5          0
```

svmon コマンドが実行されていた間に、**vmstat** コマンドが別のウィンドウで実行されました。 **vmstat** レポートは次のとおりです。

```
# vmstat 3
kthr     memory             page              faults        cpu
----- ----------- ------------------------ ------------ -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 1  5 205031 749504   0   0   0   0    0   0 1240  248 318  0  0 99  0
 2  2 151360 631310   0   0   3   3   32   0 1187 1718 641  1  1 98  0
 1  0 151366 631304   0   0   0   0    0   0 1335 2240 535  0  1 99  0
 1  0 151366 631304   0   0   0   0    0   0 1303 2434 528  1  4 95  0
 1  0 151367 631303   0   0   0   0    0   0 1331 2202 528  0  0 99  0
```

グローバル **svmon** レポートは、関連する数字を示しています。 **vmstat** コマンドの fre 欄は、**svmon** コマンドの memory free 欄と関連しています。 **vmstat** コマンドが報告するアクティブ仮想メモリー「avm」の値は、**svmon** コマンドが報告する仮想メモリーの値とほぼ同じです。

svmon コマンドと ps コマンドの出力の相関

svmon コマンドと **ps** コマンドの出力の間には、幾つかの関係があります。

例 1

以下に、**svmon** コマンドと **ps** コマンドの出力例を示します。

```
# # ps v 405528
    PID    TTY STAT  TIME PGIN  SIZE   RSS   LIM  TSIZ   TRS %CPU %MEM COMMAND
 405528  pts/0 A    43:11    1   168   172 32768     1     4 99.5  0.0 yes

(0) root @ clock16: 6.1.2.0: /
# svmon -O unit=KB,segment=category,filtercat=exclusive -P 405528
Unit: KB
--------------------------------------------------------------------------------
     Pid Command          Inuse      Pin     Pgsp  Virtual
  405528 yes                172       16        0      168

................................................................................
EXCLUSIVE segments                  Inuse      Pin     Pgsp  Virtual
                                      172       16        0      168

    Vsid      Esid Type Description              PSize  Inuse   Pin Pgsp Virtual
   554f1         f work shared library data          s     92     0    0      92
   49416         2 work process private             s     76    16    0      76
   6d49f         1 clnt code,/dev/hd2:338           s      4     0    -       -
```

上記の **ps** コマンドの出力では、SIZE が 168、RSS が 172 として表示されています。 上記の **svmon** コマンドの使用では、両方の値が表示されています。

上記に表示されている **svmon** コマンドからの出力値を、以下の式で使用すると、SIZE および RSS を計算できます。

```
SIZE = Work Process Private Memory Usage in KB + Work Shared Library Data Memory Usage in KB
RSS = SIZE + Text Code Size (Type=clnt, Description=code,)
```

上記の例の値を使用すると、次の結果になります。

```
SIZE = 92 + 76 = 168
RSS = 168 + 4 = 172
```

例 2

以下に、**svmon** コマンドと **ps** コマンドの出力例を示します。

```
# ps v 282844
    PID    TTY STAT  TIME PGIN  SIZE   RSS   LIM  TSIZ   TRS %CPU %MEM COMMAND
 282844      - A    15:49  322 24604 25280    xx   787   676  0.0  3.0 /opt/rsct/b

(0) root @ clock16: 6.1.2.0: /
# svmon -O unit=KB,segment=category,filtercat=exclusive -P 282844
Unit: KB
--------------------------------------------------------------------------------
     Pid Command          Inuse      Pin     Pgsp  Virtual
  282844 IBM.CSMAgentR    25308       16        0    24604

................................................................................
EXCLUSIVE segments                  Inuse      Pin     Pgsp  Virtual
                                    25308       16        0    24604
```

```
  Vsid      Esid Type Description              PSize  Inuse   Pin Pgsp Virtual
 2936e         2 work process private             s  23532    16    0   23532
 2d36f         f work shared library data         s   1072     0    0    1072
  1364         1 clnt code,/dev/hd2:81988         s    676     0    -       -
 154c1         - clnt /dev/hd9var:353             s     16     0    -       -
 41494         - clnt /dev/hd2:82114              s      8     0    -       -
 4d3d7         - clnt /dev/hd9var:357             s      4     0    -       -
 7935a         - clnt /dev/hd9var:307             s      0     0    -       -
 4d377         3 mmap maps 2 source(s)            s      0     0    -       -
 3934a         - clnt /dev/hd9var:300             s      0     0    -       -
```

上記の **ps** コマンドの出力では、SIZE が 24604、RSS が 25280 として表示されています。

上記に表示されている **svmon** コマンドからの出力値を、以下の式で使用すると、SIZE および RSS を計算できます。

```
SIZE = Work Process Private Memory Usage in KB + Work Shared Library Data Memory Usage in KB
RSS = SIZE + Text Code Size (Type=clnt, Description=code,)
```

上記の例の値を使用すると、次の結果になります。

```
SIZE = 23532 + 1072 = 24604
RSS = 24604 + 676 = 25280
```

最小メモリー要件の計算

プログラムの最小メモリー所要量を簡易計算する式は、次のとおりです。

```
合計メモリー・ページ (4 KB 単位) = T + ( N * ( PD + LD ) ) + F
```

ここで:

T = テキストのページの数 (すべてのユーザーが共用)

N = 同時に実行中の、このプログラムのコピーの数

PD = プロセスの専用セグメント内の作業セグメント・ページの数

LD = プロセスが使用する共用ライブラリー・データ・ページの数

F = ファイル・ページの数 (全ユーザーが共用)

結果を 4 倍して、必要な K バイト数を求めます。カーネル、カーネル・エクステンション、および共用ライブラリーのテキスト・セグメントの値も、システム内の全プロセスが共用しているものであっても、これに算入することができます。 例えば、CATIA などの一部のアプリケーションやデータベースでは、非常に大規模な共用ライブラリー・モジュールを使用します。 プロセスの単一スナップショットの統計情報のみを使用したので、式から得られた値が正しいプロセスの最小作業セットの値であるとの保証はありません。 作業セット・サイズを得るには、**rmss** コマンドなどのツールを使用するか、プロセスの寿命の間に多数のスナップショットを取って、これらのスナップショットの平均値を判別します。詳しくは、 154 ページの『rmss コマンドによるメモリー所要量の評価』を参照してください。

メモリー・リーク・プログラム

メモリー・リーク とは、繰り返しメモリーを割り当て、それを使用してから、解放するのを怠ったために発生するプログラム・エラーのことです。

対話式アプリケーションなどの、長時間実行するプログラムでのメモリー・リークは、重大な問題です。メモリーのフラグメント化や、実メモリーとページ・スペースに大部分ゴミが詰まった多数のページが累積する結果になるからです。 システムでは、1 つのプログラム内でのメモリー・リークが原因で、ページ・スペースを使い尽くしてしまうことが知られていました。

メモリー・リークは、**svmon** コマンドを使用して、作業セグメントが増え続けるプロセスを探せば、検出できます。 カーネル・セグメント内のリークは、mbuf リークによって、またはデバイス・ドライバー、カーネル・エクステンション、あるいはカーネルによっても、起きる可能性があります。 セグメントが増えているかどうかを判断するには、**-i** オプションを指定した **svmon** コマンドを使用して、プロセスまたはプロセスのグループを調べて、増え続けているセグメントがないかどうかを調べます。

問題のサブルーチンまたはコード行の識別はさらに困難で、特に AIXwindows アプリケーションでは、多数の **malloc()** コールおよび **free()** コールが生成されます。 C++ には、メモリー使用率とリークの分析/チューニングのための HeapView Debugger が用意されています。メモリー・リークの分析のためのサード・パーティー・プログラムが幾つか存在しますが、それらのプログラムでは、プログラム・ソース・コードへのアクセスが必要になります。

一部の **realloc()** サブルーチンを使用すると、実際のプログラミング・エラーではないのに、メモリー・リークと同じ影響をもたらす場合があります。 プログラムで、頻繁に **realloc()** サブルーチンを使用してデータ域のサイズを増やすと、**realloc()** サブルーチンによって解放されたストレージが、何か他の用途に使用できない場合、プロセスの作業セグメントがどんどんフラグメント化される可能性があります。

必要なくなったメモリーを解放するには、**disclaim()** システム・コールおよび **free()** コールを使用します。 **disclaim()** システム・コールは、**free()** コールの前に使用する必要があります。 これは、プログラムがすぐに終了する予定の場合は、最後の **malloc()** コールの後でメモリーを解放するために、CPU 時間を浪費します。 プログラムが終了すると、作業セグメント・データが破棄され、作業セグメントが入っていた実メモリー・ページ・フレームはフリー・リストに追加されます。次の例は、メモリー・リークが発生しているプログラムです。この専用作業セグメントの「Inuse」、「Pgspace」、および「Address Range」値は増え続けています。

```
# svmon -P 13548 -i 1 3
  Pid                          Command        Inuse        Pin      Pgsp  Virtual 64-bit Mthrd LPage
13548                           pacman         8535        2178       847     8533      N     N     N

Vsid     Esid Type  Description          LPage  Inuse    Pin  Pgsp  Virtual
    0        0 work  kernel seg               -   4375   2176   847     4375
48412        2 work  process private          -   2357      2     0     2357
6c01b        d work  shared library text      -   1790      0     0     1790
4c413        f work  shared library data      -     11      0     0       11
3040c        1 pers  code,/dev/prodlv:4097    -      2      0     -        -
ginger :svmon -P 13548 -i 1 3

  Pid                          Command        Inuse        Pin      Pgsp  Virtual 64-bit Mthrd LPage
13548                           pacman         8589       2178       847     8587      N     N     N

Vsid     Esid Type  Description          LPage  Inuse    Pin  Pgsp  Virtual
   0        0 work  kernel seg               -   4375   2176   847     4375
48412        2 work  process private          -   2411      2     0     2411
6c01b        d work  shared library text      -   1790      0     0     1790
4c413        f work  shared library data      -     11      0     0       11
3040c        1 pers  code,/dev/prodlv:4097    -      2      0     -        -

  Pid                          Command        Inuse        Pin      Pgsp  Virtual 64-bit Mthrd LPage
13548                           pacman         8599       2178       847     8597      N     N     N

 Vsid     Esid Type  Description          LPage  Inuse    Pin  Pgsp  Virtual
    0        0 work  kernel seg               -   4375   2176   847     4375
48412        2 work  process private          -   2421      2     0     2421
6c01b        d work  shared library text      -   1790      0     0     1790
4c413        f work  shared library data      -     11      0     0       11
3040c        1 pers  code,/dev/prodlv:4097    -      2      0     -        -
```

rmss コマンドによるメモリー所要量の評価

rmss コマンド (Reduced-Memory System Simulator (縮小メモリー・システム・シミュレーター)) は、メモリー・ボードを抜き出したり置き換えたりせずに、実際のマシンより小さいさまざまな実メモリーのサイズをシミュレートする手段を提供します。 その上、**rmss** コマンドは、ある範囲のメモリー・サイズにわたって、それぞれのメモリー・サイズに関して、アプリケーションの応答時間やページングの量などの、パフォーマンス統計情報を表示する機能を提供します。

rmss コマンドは、「システムが、オペレーティング・システムと与えられたアプリケーションを、容認できるレベルのパフォーマンスで実行するには、何メガバイトの実メモリーが必要か」という質問に対する答えを出すことを目的としています。 マルチユーザーの環境においては、「*X* メガバイトの実メモリーのマシンで、何人のユーザーが同時にこのアプリケーションを実行できるか」という質問に対して答えるのに役立ちます。

rmss コマンドのキャパシティー・プランニング・ツールとしての主な用途は、ワークロードが必要とするメモリーの量の判別です。 さらに、特にメモリーを増やすとパフォーマンスが低下する場合の、問題判別ツールとしても使用できます。

rmss コマンドがインストールされていて、使用可能かどうかを判別するには、次のコマンドを実行します。

```
# lslpp -lI bos.perf.tools
```

rmss コマンドによりメモリー・サイズが変更された場合、minperm と maxperm は新規パラメーターに対して調整されず、lruable ページ数はシミュレートされたメモリー・サイズに合うように変更されません。 これにより、バッファー・キャッシュが必要以上に大きくなり予期しない動作が発生する可能性があります。 その結果、システムがメモリー不足になる可能性があります。

rmss コマンドによってシミュレートされたメモリー・サイズは、オペレーティング・システムおよびその他の実行している可能性のあるプログラムが使用するメモリーを含む、マシンの実メモリーの合計サイズであることを覚えておくことが重要です。 これは、特にアプリケーション自体が使用するメモリーの量ということではありません。 **rmss** コマンドは、性能低下を引き起こす可能性があるので、root ユーザーまたはシステム・グループのメンバーだけが使用できます。

rmss コマンド

rmss コマンドは、メモリー・サイズを変更して終了する場合に使用するか、あるいは、指定されたアプリケーションをある範囲のメモリー・サイズで複数回実行し、各メモリー・サイズでのアプリケーションのパフォーマンスを記述する重要な統計情報を表示するドライバー・プログラムとして使用します。

最初の方法が便利なのは、所定のシステム・メモリー・サイズでアプリケーションがどのような動きをするかの感触をつかみたい場合、アプリケーションが複雑過ぎて単一のコマンドとして表現できない場合、あるいはアプリケーションの複数インスタンスを実行したい場合です。 2 番目の方法は、実行可能プログラムまたはシェル・スクリプト・ファイルとして起動できるアプリケーションの場合に適しています。

rmss コマンドの **-c**、**-p**、および **-r** フラグ**:**

rmss コマンドの **-c**、**-p** および **-r** フラグの利点は、1 つの実行可能プログラムやシェル・スクリプト・ファイルの形で表すことができない、複雑なアプリケーションを実験できるということです。 これに対して、**-c**、**-p**、および **-r** オプションの欠点は、ユーザーが独自のパフォーマンス測定を行うように強制されるという点です。 幸い、**vmstat -s** コマンドを使用すると、アプリケーションの実行中に発生したページング・スペースのアクティビティーを測定することができます。

vmstat -s コマンドを実行することによって、アプリケーションを実行してから、その後に **vmstat -s** を再び実行すれば、前のページング・スペースのページインの数を後のページインの数から引いて、プログラムの実行中に発生したページング・スペースのページインの数を決定することができます。 さらに、プログラムの時間を測定しておき、ページング・スペースのページインの回数をプログラムの経過実行時間で割れば、ページング・スペースの平均ページイン率を求めることができます。

次の 2 つの理由から、それぞれのメモリー・サイズでアプリケーションを複数回実行することも重要です。

- メモリー・サイズを変更するとき、**rmss** コマンドは通常、多くのメモリーをクリアする。 したがって、メモリー・サイズの変更後に最初にアプリケーションを実行したとき、実行時間の相当部分が、アプリケーションが実メモリーにファイルを読み込むためである可能性があります。 しかし、ファイルはアプリケーションの終了後もメモリー内に残すことができるので、その後アプリケーションを実行するときには、経過時間がかなり短くなる可能性があります。
- そのメモリー・サイズでのアプリケーションの平均パフォーマンスの感触をつかむため。 アプリケーションの実行のたびにシステム状態を重複させるのは、不可能です。 このため、アプリケーションのパフォーマンスは実行のたびにかなり変化する可能性があります。

要約すると、**rmss** コマンドを起動する場合は、次のステップを望ましい方法として考慮してください。

```
while there are interesting memory sizes to investigate:
  {
  change to an interesting memory size using rmss -c;
  run the application once as a warm-up;
  for a couple of iterations:
    {
    use vmstat -s to get the "before" value of paging-space page ins;
    run the application, while timing it;
    use vmstat -s to get the "after" value of paging-space page ins;
    subtract the "before" value from the "after" value to get the
       number of page ins that occurred while the application ran;
    divide the number of paging-space page ins by the response time
       to get the paging-space page-in rate;
    }
  }
run rmss -r to restore the system to normal memory size (or reboot)
```

(after - before) ページ入出力数の計算は、 39 ページの『ディスクまたはメモリー関連の問題』に説明されている、**vmstatit** スクリプトを使用して、自動化することができます。

メモリー・サイズ変更*:*

メモリー・サイズを変更して終了する場合は、**rmss** コマンドの **-c** フラグを使用します。

メモリー・サイズを 128 MB に変更するには、例えば、以下のコマンドを使用します。

```
# rmss -c 128
```

メモリー・サイズは、メガバイト単位の整数または 10 進の小数 (例えば、128.25) です。 さらに、サイズは、8 MB とマシンの物理的実メモリーの量の間でなければなりません。 ハードウェアとソフトウェアの構成によっては、カーネルなどの固有のシステム構造体のサイズに制約があるために、**rmss** コマンドでメモリー・サイズを小さいサイズに変更できない場合があります。 **rmss** コマンドで所定のメモリー・サイズに変更できない場合は、エラー・メッセージが表示されます。

rmss コマンドは、VMM が維持する空きフレームのリストから空きページ・フレームをスチールすることによって、システムの実効メモリー・サイズを減らします。 スチールされたフレームは、使用できないフ

レームのプールに保持され、実効メモリー・サイズを増やすときに、空きフレーム・リストに戻されます。また、**rmss** コマンドは、メモリーの実効サイズに比例して保持する必要がある一定のシステム変数とデータ構造体を動的に調整します。

メモリー・サイズを変更するには、多少時間がかかる可能性があります (最大 15 秒から 20 秒)。 一般に、縮小するメモリー・サイズが大きくなるほど、**rmss** コマンドが完了するまでの時間が長くなります。成功すると、**rmss** コマンドが次のメッセージを戻します。

```
Simulated memory size changed to  128.00 Mb.
```

現在のメモリー・サイズを表示するには、**-p** フラグを次のように使用します。

```
# rmss -p
```

rmss の出力は次のようになります。

```
Simulated memory size is  128.00 Mb.
```

最後に、メモリー・サイズをマシンの実メモリー・サイズにリセットしたい場合は、**-r** フラグを次のように使用します。

```
# rmss -r
```

現在シミュレートしているメモリー・サイズに関係なく、**-r** フラグを使用すると、メモリー・サイズがマシンの物理的実メモリー・サイズにセットされます。

この例は 256 MB のマシンで実行されているので、**rmss** コマンドは次のように応答します。

```
Simulated memory size changed to  256.00 Mb.
```

注**: rmss** コマンドは、使用可能実メモリーを報告します。 不良メモリーまたは使用中のメモリーを含むマシンでは、**rmss** コマンドは実メモリーの量を、物理実メモリーから不良メモリーまたはシステムが使用しているメモリーを引いた量として報告します。 例えば、**rmss -r** コマンドは次のように報告します。

```
Simulated memory size changed to 79.9062 Mb.
```

これは、一部のページが不良とマークされているか、またはデバイスが自分で使用するために一部のページを予約しているのでユーザーが使用できなくなっていることの結果である可能性があります。

rmss コマンドの使用によるメモリー・サイズのある範囲でのアプリケーションの実行**:**

ドライバー・プログラムとしては、**rmss** コマンドは、指定されたアプリケーションをある範囲のメモリー・サイズにわたって実行し、各メモリー・サイズでのアプリケーションのパフォーマンスを記述する統計情報を表示します。

rmss コマンドの **-s**、**-f**、**-d**、**-n**、および **-o** の各フラグは、**rmss** コマンドをドライバー・プログラムとして起動するために組み合わせて使用されます。 この **rmss** コマンドの起動スタイルの構文は、次のとおりです。

```
rmss [ -s smemsize ] [ -f fmemsize ] [ -d memdelta ]
     [ -n numiterations ] [ -o outputfile ] command
```

次のフラグは、それぞれ後で詳しく説明します。 **-s**、**-f**、および **-d** フラグは、メモリー・サイズの範囲の指定に使用されます。

-n このフラグは、各メモリー・サイズについてコマンドを実行および測定する回数を指定するために使用されます。

-o このフラグは、**rmss** レポートを書き込むファイルの指定に使用されます。コマンドは、各メモリー・サイズで実行および測定したいアプリケーションです。

-s このフラグは、開始サイズを指定します。

-f このフラグは、最終サイズを指定します。

-d このフラグは、サイズ間の差を指定します。

すべての値は整数または 10 進の小数で、メガバイト単位です。 例えば、256、224、192、160 および 128 MB のサイズでコマンドを実行および測定したい場合は、次の組み合わせを使用します。

```
-s 256 -f 128 -d 32
```

同様に、128、160、192、224、および 256 MB のサイズでコマンドを実行および測定したい場合は、次の組み合わせを使用します。

```
-s 128 -f 256 -d 32
```

-s フラグを省略した場合、**rmss** コマンドはマシンの実際のメモリー・サイズから開始します。 **-f** フラグを省略した場合、**rmss** コマンドは 8 MB で終了します。 **-d** フラグを省略した場合、各メモリー・サイズ間はデフォルトで 8 MB です。

-s、**-f**、および **-d** のフラグにどの値を選択するべきでしょうか。簡単な選択方法は、測定するアプリケーションを実行しようと考えているシステムのメモリー・サイズをカバーすることです。 しかし、増分を 8 MB 未満にすると便利です。所定のサイズに決めるときに、どれだけのスペースになるかを見積もることができるからです。 例えば、所定のアプリケーションが 120 MB ではスラッシングするが、128 MB ではページインなしで実行する場合、120 MB から 128 MB までの範囲のどこでアプリケーションがスラッシングを始めるということが分かれば役に立ちます。 127 MB でスラッシングが始まれば、128 MB より大きいメモリーのシステムを構成することを考慮するか、またはもっとスペースができるようにアプリケーションを変更してみる必要があります。 一方、スラッシングが 121 MB で始まる場合、128 MB のマシンで十分なスペースがあることが分かります。

-n フラグは、各メモリー・サイズでコマンドを実行および測定する回数を指定するために使用されます。コマンドを指定された回数だけ実行および測定した後で、**rmss** コマンドは、そのメモリー・サイズでのアプリケーションの平均パフォーマンスを記述した統計情報を表示します。各メモリー・サイズでコマンドを 3 回実行するには、次のコマンドを使用します。

```
-n 3
```

-n フラグを省略した場合、**rmss** コマンドは、初期化時に、合計実行時間を 10 秒まで累積するにはアプリケーションを何回実行すればよいかを判別します。 **rmss** コマンドは、短時間実行するプログラムのパフォーマンス統計情報が、デーモンなどの外部の影響によって大きくゆがめられないようにするために、これを行います。

注**:** 非常に短いプログラムを測定する場合、 10 秒の CPU 時間を累積するために必要な反復回数は、非常に大きくなる可能性があります。 プログラムは、実行するたびに最低でも約 2 経過秒の **rmss** オーバーヘッドがかかるので、短いプログラムについては、**-n** パラメーターを明示的に指定してください。

-n フラグについてはどの値を使用するのがよいのでしょうか。 アプリケーションの実行時間が 10 秒をかなり超える場合は、**-n 1** と指定して、各メモリー・サイズについてコマンドを 2 回実行し、測定は 1 回だけ行うこともできます。 **-n** フラグを使用することの利点は、**rmss** コマンドが、初期化時にプログラムを何回実行するかの判別に時間を費やす必要がないので、早く終了するということです。 これは、測定されるコマンドが長時間かかり、対話式であるときに、特に価値があります。

rmss コマンドが、コマンドを実行および測定する前に、常にウォームアップのために各メモリー・サイズごとに 1 回ずつコマンドを実行することに注意することが重要です。 ウォームアップは、アプリケーションがまだメモリーに入っていないときに起こる入出力を避けるために必要です。 このような入出力はパフォーマンスには影響しませんが、必ずしも実メモリーの不足によるものとは限りません。 ウォームアップの実行は、**-n** フラグで指定される反復回数には含まれません。

-o フラグは、**rmss** レポートを書き込むファイルの指定に使用されます。 **-o** フラグを省略すると、レポートは rmss.out ファイルに書き込まれます。

最後に、**command** は測定するアプリケーションの指定に使用されます。 これは、実行可能プログラムまたはシェル・スクリプトで、コマンド・ライン引数は付けても付けなくても構いません。 ただし、コマンドの形式には幾つかの制限があります。 最初に、入力または出力のリダイレクトは含めることができません (例えば、 **foo > output** または **foo < input**)。 これは、**rmss** コマンドが、コマンド名の右側のものをすべてコマンドに対する引数として扱うためです。 リダイレクトするには、コマンドをシェル・スクリプト・ファイルに置きます。

通常、**rmss** の出力を特定のファイルに保管したい場合は、 **-o** オプションを使用します。 **rmss** コマンドの標準出力をリダイレクトして (例えば、それを既存ファイルの最後に連結して)、次に Korn シェルを使用して **rmss** の起動を括弧で囲みたい場合は、次のようにします。

```
# (rmss -s 24 -f 8 foo) >> output
```

rmss コマンド結果の変換処理

rmss コマンドは有益な情報を生成します。

『foo プログラムに関して生成されたレポート』 トピックの例は、実際のアプリケーション・プログラムで **rmss** コマンドを実行して作成されました。ただし、プログラムの名前は匿名にするため foo に変更されています。 レポート生成のための実際のコマンドは、次のとおりです。

```
# rmss -s 16 -f 8 -d 1 -n 1 -o rmss.out foo
```

foo プログラムに関して生成されたレポート:

rmss コマンドは **foo** プログラムに関するレポートを作成します。

```
Hostname:  aixhost1.austin.ibm.com
Real memory size:   16.00 Mb
Time of day:  Thu Mar 18 19:04:04 2004
Command:  foo

Simulated memory size initialized to  16.00 Mb.

Number of iterations per memory size = 1 warm-up + 1 measured = 2.

Memory size  Avg. Pageins  Avg. Response Time    Avg. Pagein Rate
(megabytes)                (sec.)                (pageins / sec.)
--------------------------------------------------------------------
16.00           115.0           123.9                 0.9
15.00           112.0           125.1                 0.9
14.00           179.0           126.2                 1.4
13.00            81.0           125.7                 0.6
12.00           403.0           132.0                 3.1
11.00           855.0           141.5                 6.0
10.00          1161.0           146.8                 7.9
9.00           1529.0           161.3                 9.5
8.00           2931.0           202.5                14.5
```

レポートは 4 つの欄で構成されています。 左端の欄はメモリー・サイズを示し、一方、「*Avg. Pageins*」欄は、アプリケーションがそのメモリー・サイズで実行されたときに発生したページインの平均数を示します。 「*Avg. Pageins*」欄が、コード、データ、およびファイルの、アプリケーションの実行時に完了したすべてのプログラムからの読み取りを含む、すべてのページイン操作を表していることに注意することが重要です。 「*Avg. Response Time*」欄は、アプリケーションの完了までの平均時間を示し、「*Avg. Pagein Rate*」欄は、平均ページイン率を示します。

「*Avg. Pagein Rate*」欄に注目してください。 16 MB から 13 MB までは、ページイン率は比較的小さいです (< 1.5 ページイン/秒)。 ただし、13 MB から 8 MB までは、ページインが最初は徐々に大きくなり、その後 8 MB に達すると急速に大きくなります。 「*Avg. Response Time*」欄もこれと似た状態で、最初は比較的平らですが、次第に増加していって、最後にメモリー・サイズが 8 MB に達したところで急速に増加しています。

ここで、ページイン率は、メモリー・サイズが 14 MB (1.4 ページイン/秒) から 13 MB (0.6 ページイン/秒) に変化するときに、実際には減少しています。 これがアラームの原因ではありません。 実際のシステムでは、結果が完全に滑らかになると期待することはできません。 重要な点は、ページイン率が 14 MB と 13 MB の両方で比較的低いということです。

最後に、レポートから 2 つの結論を導き出すことができます。 まず、アプリケーションのパフォーマンスが 8MB で受け入れ難いと思われた場合 (おそらくそうでしょうが)、メモリーを追加するとパフォーマンスがかなり改善されるはずです。 応答時間が 16 MB の約 124 秒から 8 MB では 202 秒に、63% も増加していることに注意してください。 一方、16 MB でのパフォーマンスが受け入れ難いと思われる場合は、メモリーを追加しても、16 MB でページインによってプログラムが目に見えて遅くなるわけではないので、パフォーマンスが大きく改善されることはありません。

16 MB のリモート・コピーのレポート**:**

次の例は、リモート (サーバー) マシンから NFS を介して 16 MB のファイルをコピーしたコマンドについて、(クライアント・マシンで) **rmss** コマンドを実行して、生成されたレポートを示しています。

```
Hostname:  aixhost2.austin.ibm.com
Real memory size:   48.00 Mb
Time of day:  Mon Mar 22 18:16:42 2004
Command:  cp /mnt/a16Mfile /dev/null

Simulated memory size initialized to  48.00 Mb.

Number of iterations per memory size = 1 warm-up + 4 measured = 5.

Memory size   Avg. Pageins   Avg. Response Time  Avg. Pagein Rate
(megabytes)                     (sec.)           (pageins / sec.)
--------------------------------------------------------------------
48.00              0.0            2.7                   0.0
40.00              0.0            2.7                   0.0
32.00              0.0            2.7                   0.0
24.00              1520.8        26.9                  56.6
16.00              4104.2        67.5                  60.8
8.00               4106.8        66.9                  61.4
```

このレポートの応答時間とページイン率は、比較的低く始まり、24 MB のメモリー・サイズで急激に増加し、16 MB と 8 MB で水平状態に達します。 このレポートは、**rmss** コマンドを使用するときは、幅広いメモリー・サイズを選択することが重要であることを示しています。 このユーザーが 24 MB から 8 MB の範囲のメモリー・サイズだけを見ていたとしたら、ページインなしでアプリケーションに対応できる、十分なメモリーを持ったシステムを構成するチャンスを逃していたかもしれません。

-s、**-f**、**-d**、**-n**、および **-o** フラグの使用のヒント**:**

rmss コマンドの便利な機能の 1 つは、このように使用した場合、出力ファイルに書き込まれているレポートを破棄せずに割り込みキー (デフォルトでは **Ctrl** + **C**) で終了することができます。 この機能により、出力ファイルへのレポートの書き込みに加えて、**rmss** コマンドに、メモリー・サイズをマシンの物理メモリー・サイズにリセットさせることになります。

ログアウトした後でも、**nohup** コマンドを使用すれば、 **rmss** コマンドをバックグラウンドで実行することができます。それには次のように、**rmss** コマンドの前に **nohup** コマンドを置き、コマンド全体の後に & (アンパーサンド) を付けます。

```
# nohup rmss -s 48 -f 8 -o foo.out foo &
```

rmss コマンドの使用時に考慮すべきガイドライン

どちらの **rmss** 起動スタイルを使用するにしても、できるだけエンド・ユーザー環境に近い環境を実現することが重要です。

例えば、同じ型の CPU、同じ型のディスク、同じ型のネットワークを使用していますか。 ユーザーは、リモート・ノードから NFS または他の分散ファイルシステムを介して、アプリケーション・ファイルをマウントすることがありますか。 この最後の点は特に重要です。リモート・ファイルのページは、VMM によって、ローカル・ファイルのページとは別に扱われるからです。

同様に、希望のシステム構成または測定するアプリケーションと関連がないシステム・アクティビティーは、除去するのが最良です。例えば、**rmss** コマンドを実行するマシンでは、測定するワークロードの一部を担っているユーザー以外には作業させないようにします。

注**: rmss** コマンドは、同時に複数起動することはできません。

rmss コマンドの実行がすべて完了したときに、システムをシャットダウンしてリブートするのが最適です。 この作業で、**rmss** コマンドがシステムに対して行ったすべての変更が削除され、 VMM メモリー・ロード制御パラメーターが通常の設定に復元されます。

schedo コマンドによる VMM メモリー・ロード制御チューニング

schedo コマンドによって、root ユーザーは、スラッシングの判別に使用される基準、どのプロセスを中断するかの決定に使用される基準、スラッシングの終了後にプロセスを再活動化するまでの待ち時間の長さ、中断を免除されるプロセスの最小数に影響を与えるか、または値をデフォルトにリセットすることができます。

VMM メモリー・ロード制御機能 (55 ページの『VMM メモリー・ロード制御機能』で説明) は、過負荷のシステムをスラッシングから保護します。

以前のバージョンのオペレーティング・システムでは、多数のプロセスが同時にシステムをヒットすると、メモリーがオーバーコミットになってスラッシングが起こり、パフォーマンスが急速に低下します。 メモリー・ロード制御のメカニズムは、スラッシングを検出するために開発されました。 ある種のパラメーターは、ロード制御のメカニズムに影響を与えます。

schedo コマンドがインストールされていて、使用可能かどうかを判別するには、次のコマンドを実行します。

```
# lslpp -lI bos.perf.tune
```

メモリー・ロード制御のチューニング

メモリー・ロード制御は、システムでスラッシングが起こる可能性のある、まれにある負荷のピークを平滑化することを目的としています。

メモリー・ロード制御は、マルチプログラミングとスループットのトレードオフということであり、 RAM が小さすぎて通常のワークロードを処理できないようなシステム構成で連続運用することが目的ではありません。 この設計はバッチ・ジョブ用であり、プロセスによって区別することはありません。 AIX ワークロード・マネージャーは、重要なタスクの保護のための優れたソリューションを提供します。

根本的な RAM 不足に対する正しいソリューションとは、RAM を追加することであり、メモリー・ロード制御を使用して、応答時間とメモリーのトレードオフを試みる実験をすることではありません。 メモリー・ロード制御機能のチューニングが本当に必要な状況とは、RAM が十分あり、デフォルト未満の値が選択されてはいない状況です。 例えば、デフォルトの値が小さすぎる構成です。

ワークロードは安定しているが、デフォルト・パラメーターがそのワークロードに不適切だと確信できる場合でなければ、メモリー・ロード制御パラメーターの設定を変更するべきではありません。

システム出荷時のデフォルトのパラメーター設定は、変更しない限り常に有効です。 これらのパラメーターのデフォルト値は、広範なワークロードの「フェイルセーフ」として動作するように選択されています。変更したパラメーターが有効なのは、次回のシステム・ブートまでです。 メモリー・ロード制御のチューニング・アクティビティーはすべて、root ユーザーが行う必要があります。 システム管理者は、**schedo** コマンドを使用してパラメーターを変更し、特定のワークロードにあわせてアルゴリズムをチューニングするか、またはすべてを使用不可にすることができます。

以下は、**schedo** コマンドによって現在のパラメーター値を表示する例です。

```
# schedo -a
           v_repage_hi = 0
         v_repage_proc = 4
            v_sec_wait = 1
         v_min_process = 2
         v_exempt_secs = 2
              pacefork = 10
               sched_D = 16
               sched_R = 16
             timeslice = 1
               maxspin = 1
              %usDelta = 100
          affinity_lim = n/a
idle_migration_barrier = n/a
      fixed_pri_global = n/a
         big_tick_size = 1
             force_grq = n/a
```

最初の 5 つのパラメーターは、メモリー・ロード制御アルゴリズムのしきい値を指定します。 これらのパラメーターは、アルゴリズムの率としきい値を設定します。 *v_repage_proc*、*v_min_process*、*v_sec_wait*、*v_exempt_secs* の値は、アルゴリズムが RAM のオーバーコミットを示す場合に使用されます。 それ以外の場合、これらの値は無視されます。 メモリー・ロード制御が使用不可の場合、これらの文字は使用されません。

チューニングの実験後、メモリー・ロード制御は、**schedo -D** コマンドの実行によって、デフォルトの特性にリセットすることができます。

v_repage_hi パラメーター**:**

v_repage_hi パラメーターは、メモリーのオーバーコミットを定義するしきい値を制御します。 メモリー・ロード制御は、任意の 1 秒間にこのしきい値を超えると、プロセスを中断しようとします。

しきい値は、2 つの直接の測定、最後の 1 秒間にページング・スペースに書き込まれたページの数 (po) と最後の 1 秒間のページ・スチール発生の数 (fr) の間の関係です。 これらの値は両方とも **vmstat** の出力に表示されます。 ページ書き込みの数は、通常はページ・スチールの数よりはるかに小さくなります。 次の場合は、メモリーはオーバーコミットされています。

```
po/fr > 1/v_repage_hi or po*v_repage_hi > fr
```

schedo -o v_repage_hi=0 コマンドは、メモリー・ロード制御を事実上使用不可にします。 システムに少なくとも 128 MB のメモリーがある場合、デフォルト値は 0、それ以外の場合のデフォルト値は 6 です。最低でも 128 MB の RAM がある場合、通常の VMM アルゴリズムは、平均して、メモリー・ロード制御を使用した場合より効果的に、スラッシング条件を訂正します。

一部の特殊な状況では、初めからメモリー・ロード制御を使用不可にするのが適切な場合があります。 例えば、マルチユーザー・ワークロードをシミュレートするために、タイムアウト・フィーチャー付きの端末エミュレーターを使用している場合、メモリー・ロード制御が介入すると、タイムアウト・フィーチャーによって、プロセスが kill されてしまうほど、応答が遅れる場合があります。 他の例としては、**rmss** コマンドを使用して、メモリー・サイズを削減した場合の影響を調べている場合は、測定による干渉をさけるためにメモリー・ロード制御を使用不可にしてください。

メモリー・ロード制御を使用不可にすると、スラッシング状態が減るというより増える場合は (それにしたがって、応答性も悪くなる)、メモリー・ロード制御がシステムでアクティブかつ有効な役割を演じていることになります。 したがって、メモリー・ロード制御パラメーターをチューニングすると、パフォーマンスが向上するか、さもなければ、RAM の追加が必要になる場合があります。

v_repage_hi の値を低くすると、スラッシング検出のしきい値が上がります。つまり、プロセスが中断状態になる前にシステムがより早くスラッシングに入ることになります。 システム構成とは関係なく、po/fr の値が低いと、スラッシングが起こりにくくなります。

しきい値を 4 に変更するには、以下を入力します。

```
# schedo -o v_repage_hi=4
```

このようにして、アルゴリズムがプロセスの中断を開始する前に、システムがスラッシングに入るように制御できます。

v_repage_proc パラメーター**:**

v_repage_proc パラメーターは、プロセスが中断の対象となるかどうかの判別と、すべてのプロセスについて維持されている 2 つの測定値、再ページの数 (*r*) と、最後の 1 秒間にプロセスが累積したページ・フォールトの数 (*f*) の比率のしきい値の設定に使用されます。

再ページのページ・フォールトに対する比率が高いのは、個々のプロセスがスラッシングしていることを意味しています。 プロセスは次の条件が成立する場合、プロセスは中断 (これはスラッシングまたは全体的スラッシングの一助になる) の対象になると見なされます。

```
r/f > 1/v_repage_proc or r*v_repage_proc > f
```

v_repage_proc のデフォルト値は 4 で、これは、最後の 1 秒における再ページのページ・フォールトに対する部分が 25% を超えると、プロセスがスラッシングである (そして中断の候補である) と見なされるこ

とを意味します。 *v_repage_proc* の値を低くすると、プロセスが中断の対象になる前に、個々のプロセスがスラッシングに入ることが許される度合いが高くなります。

メモリー・ロード制御によるプロセスの中断を使用不可にするには、下記を使用します。

```
# schedo -o v_repage_proc=0
```

固定優先順位のプロセスとカーネル・プロセスは、中断を免除されていることに注意してください。

v_min_process パラメーター**:**

v_min_process パラメーターは、マルチプログラミングの程度の下限を、アクティブ・プロセスの数として定義することで決定します。 アクティブ・プロセスとは、実行可能で、ページ入出力を待っているプロセスです。イベント待ちのプロセスと中断状態のプロセスはアクティブとは見なされず、また待ちプロセスもアクティブとは見なされません。

マルチプログラミングの最小レベル *v_min_process* パラメーターを設定すると、 *v_min_process* 個のプロセスが中断されるのを効果的に防ぐことができます。 システム管理者が、よいパフォーマンスを実現するには、少なくとも 10 個のプロセスが常に RAM に常駐してアクティブである必要があると考えており、メモリー・ロード制御によるプロセスの中断が活発過ぎると疑っているとします。 **schedo -o v_min_process=10** コマンドが実行された場合、システムは、10 個未満のプロセスでメモリーの競合が行われているかぎり、プロセスを中断することはありません。 *v_min_process* パラメーターには、下記のプロセスはカウントされていません。

- カーネル・プロセス
- **plock()** システム・コールによって、RAM 内で固定されているプロセス
- 優先順位の値が 60 未満の固定優先順位プロセス
- イベント待ちのプロセス

システム・デフォルトの **v_min_process=2** では、カーネル、すべての固定プロセス、および 2 つのユーザー・プロセスは、常に RAM を競合するプロセスの集合に含まれます。

v_min_process=2 はデスクトップのシングルユーザー構成には適していますが、規模の大きい、マルチユーザー、または大量の RAM を搭載したサーバー構成には、ほとんどの場合小さすぎます。

インストールするシステムが 32 MB より大きく、128 MB より小さい場合、一度に 5 つ以上のアクティブ・ユーザーをサポートすると予想される場合、 VMM メモリー・ロード・メカニズムのマルチプログラミングの最低レベルを上げることを考慮してください。

例えば、控えめに見積もって、最もメモリー集中のアプリケーションを 4 つ同時に実行する必要があり、少なくとも 16 MB をオペレーティング・システム用に、実メモリーの 25% をファイル・ページ用に残すとすれば、次のコマンドで、マルチプログラミングの最低レベルをデフォルトの 2 から 4 へ上げることができます。

```
# schedo -o v_min_process=4
```

これらのシステムでは、*v_min_process* パラメーターを 4 または 6 に設定すると、最高のパフォーマンスが得られる可能性があります。 *v_min_process* の値を下げることは可能ですが、ユーザー・プロセスを 1 つしかアクティブにできないことが時々あります。

スラッシングが発生するアプリケーションのメモリー所要量が分かっている場合は、適切な *v_min_process* の値を選択することができます。 スラッシングは、サイズが *M* のアプリケーションのインスタンスが多数あるために発生しているとします。システム・メモリー・サイズが *N* とすると、*v_min_process* パラメ

ーターは *N*/*M* に近い値に設定します。 *v_min_process* パラメーターの設定が低すぎると、同時にアクティブになれるプロセスの数が不必要に制限されることになります。

v_sec_wait パラメーター**:**

v_sec_wait パラメーターは、中断状態のプロセスが再活動化されるために、po/fr の比率が、継続して 1/v_repage_hi 未満でなければならない期間の、1 秒間隔の数を制御します。

デフォルト値の 1 秒は、可能な最小値、つまりゼロに近い値です。 値 1 秒では、1 秒の安全期間が始まるとすぐに、積極的にプロセスを再活動化しようとします。 *v_sec_wait* を大きくすると、アクティブなプロセスがないためにプロセッサーがアイドルになっている間の、中断状態のプロセスの応答時間が悪くなります。

待ち時間を、2 秒後にプロセスを再活動化するように変更するには、次のように入力します。

```
# schedo -o v_sec_wait=2
```

v_exempt_secs パラメーター**:**

中断状態のプロセスは、再活動化されるたびに、*v_exempt_secs* 経過秒の期間だけ中断を免除されます。 これにより、中断状態のプロセスのページをページインするディスク入出力の高いコストを、進行のための妥当な機会にすることができます。

v_exempt_secs のデフォルト値は 2 秒です。

このパラメーターを変更するには、次のように入力します。

```
# schedo -o v_exempt_secs=1
```

多くのメモリーを使用するが、約 *T* 秒間しか実行しないアプリケーションで、時々スラッシングが起こるとします。 *v_exempt_secs* パラメーターをデフォルトのシステム設定である 2 秒にすると、ビジーなシステムではこのアプリケーションが *T*/2 回スワッピング・インおよびアウトすることになります。 この場合、*v_exempt_secs* パラメーターをもっと長い時間に再設定すると、このアプリケーションの進行を助けることができます。 この問題のアプリケーションを無理やり押し進めてしまえば、システム・パフォーマンスが改善されます。

VMM ページ置換のチューニング

メモリー管理アルゴリズムでは、フリー・リストのサイズと指定されたバウンダリー内の永続セグメント・ページが占める実メモリーのパーセンテージを保持しようとします。

このバウンダリーは、 50 ページの『実メモリーの管理』で検討されていますが、**vmo** コマンドによって変更でき、root ユーザーだけが実行できます。 このツールによる変更は、次回のシステムのリブートまで有効です。 **vmo** コマンドがインストールされていて、使用可能かどうかを判別するには、次のコマンドを実行します。

```
# lslpp -lI bos.perf.tune
```

vmo コマンドを **-a** オプション付きで実行すると、現在のパラメーター設定が表示されます。

注**: vmo** コマンドは、自己文書化コマンドです。以下に示す出力例とは異なる出力が得られる場合があります。

```
# vmo -a
               ame_cpus_per_pool = n/a
                 ame_maxfree_mem = n/a
             ame_min_ucpool_size = n/a
```

```
               ame_minfree_mem = n/a
               ams_loan_policy = n/a
  enhanced_affinity_affin_time = 1
enhanced_affinity_vmpool_limit = 10
                esid_allocator = 1
           force_relalias_lite = 0
             kernel_heap_psize = 65536
                  lgpg_regions = 0
                     lgpg_size = 0
              low_ps_handling = 1
                       maxfree = 1088
                       maxperm = 843105
                        maxpin = 953840
                       maxpin% = 90
                memory_frames = 1048576
                memplace_data = 0
          memplace_mapped_file = 0
        memplace_shm_anonymous = 0
            memplace_shm_named = 0
                 memplace_stack = 0
                memplace_text = 0
        memplace_unmapped_file = 0
                       minfree = 960
                       minperm = 28103
                      minperm% = 3
                  msem_nlocks = 0
                    nokilluid = 0
                      npskill = 1024
                      npswarn = 4096
           num_locks_per_semid = 1
                     numpsblks = 131072
                    pgz_lpgrow = 2
                      pgz_mode = 2
               pinnable_frames = 781272
           relalias_percentage = 0
                         scrub = 0
                thrpgio_inval = 1024
               thrpgio_npages = 1024
                     v_pinshm = 0
                vm_cpu_thresh = 0
                 vm_mmap_bmap = 1
              vmm_default_pspa = 0
                vmm_klock_mode = 2
             wlm_memlimit_nonpg = 1
```

minfree および maxfree パラメーターの値

フリー・リストの目的は、終了するプロセスが解放した実メモリーのページ・フレームの状況を常に把握し、ページ・スチールとそれに伴う入出力の完了を強制的に待たせることなく、要求元に即時にページ・フレームを提供できるようにすることです。

minfree 制限は、それより下がると、フリー・リストを補充するためのページ・スチールが開始されるフリー・リスト・サイズを指定します。 *maxfree* パラメーターは、それを超えるとスチール処理が終了するサイズです。ページ・スチールを開始するには *minfree* 値を使用します。ページ・スチールが開始されるのは、永続ページ数が、パラメーターの *maxfree* と *minfree* の値の差に等しいかそれより小さい場合、またはクライアント・ページ数が、パラメーターの *maxclient* と *minfree* の値の差に等しいかそれより小さい場合です。

これらの制限のチューニングの目的は、以下を保証することです。

- 応答時間が重要な目標であるアクティビティーは、常に必要なページ・フレームをフリー・リストから入手できる。

- フリー・リストの拡張のために早すぎるページ・スチールが行われ、その結果、システムで不必要に高レベルの入出力が発生することがない。

パラメーター *minfree* と *maxfree* のデフォルト値は、マシンのメモリー・サイズによって決まります。 JFS 使用時は、パラメーター *maxfree* と *minfree* の差は、常に、パラメーター *maxpgahead* の値と等しいかそれ以上でなければなりません。 拡張 JFS の場合、パラメーター *maxfree* と *minfree* の差は、常に、パラメーター *j2_maxPageReadAhead* の値と等しいかそれ以上でなければなりません。 JFS と拡張 JFS の両方を使用している場合、パラメーター *minfree* の値を、2 つのファイルシステムの大きい方の pageahead 値より大きいか等しい数値に設定する必要があります。

メモリー・プールが複数個あると、パラメーター *minfree* と *maxfree* の値は異なります。 メモリー・プールは、大量の RAM を搭載した MP システム用に導入されました。 各メモリー・プールには、独自の *minfree* および *maxfree* 値があります。 AIX の旧バージョンでは、**vmo** コマンドによって表示される *minfree* および *maxfree* 値は、すべてのメモリー・プールの *minfree* 値と *maxfree* 値の合計です。 **vmo** コマンドによって表示される値はメモリー・プールごとになります。

minfree の適切なサイズを調べるための、正確ではないがより包括的なツールは、**vmstat** コマンドです。 以下は、*minfree* 値に達しているシステムに関する **vmstat** コマンドの出力の一部です。

```
# vmstat 1
kthr     memory             page              faults        cpu
----- ----------- ------------------------ ------------ -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs   us sy id wa
 2  0 70668   414   0   0   0   0    0   0 178 7364  257  35 14  0 51
 1  0 70669   755   0   0   0   0    0   0 196 19119 272  40 20  0 41
 1  0 70704   707   0   0   0   0    0   0 190 8506  272  37  8  0 55
 1  0 70670   725   0   0   0   0    0   0 205 8821  313  41 10  0 49
 6  4 73362   123   0   5  36 313 1646   0 361 16256 863  47 53  0  0
 5  3 73547   126   0   6  26 152  614   0 324 18243 1248 39 61  0  0
 4  4 73591   124   0   3  11  90  372   0 307 19741 1287 39 61  0  0
 6  4 73540   127   0   4  30 122  358   0 340 20097 970  44 56  0  0
 8  3 73825   116   0  18  22 220  781   0 324 16012 934  51 49  0  0
 8  4 74309    26   0  45  62 291 1079   0 352 14674 972  44 56  0  0
 2  9 75322     0   0  41  87 283  943   0 403 16950 1071 44 56  0  0
 5  7 75020    74   0  23 119 410 1611   0 353 15908 854  49 51  0  0
```

上記の出力例では、*minfree* 値 (120) に絶えず達していることが確認できます。 したがって、ページ置換が発生し、この場合は、ある時点でフリー・リストが 0 に達しています。 このような状態が発生すると、フリー・フレームを必要とするスレッドはブロックされ、ページ置換がいくつかのページを解放するまで実行できなくなります。 このような状態を避けるために、 *minfree* 値と *maxfree* 値を増やすことを検討してください。

メモリー・プールごとに少なくとも 1000 ページを常にフリーにするには、次のコマンドを実行してください。

```
# vmo -o minfree=1000 -o maxfree=1008
```

これを永続変更にするには、**-p** フラグを組み込んでください。

```
# vmo -o minfree=1000 -o maxfree=1008 -p
```

パラメーター *minfree* のデフォルト値は、メモリー・プールごとに 960 まで増加し、パラメーター *maxfree* のデフォルト値は、メモリー・プールごとに 1088 まで増加します。

リスト・ベースの LRU

LRU アルゴリズムはリストを使用します。 AIX の旧バージョンでは、ページ・フレーム・テーブル方式も使用できました。リスト・ベース・アルゴリズムでは、各タイプのセグメントをスキャンするためにページ・リストが提供されます。

以下は、セグメントのタイプのリストです。

- 処理中
- 永続
- クライアント
- 圧縮

WLM が使用可能であれば、クラスのリストもあります。

lrubucket パラメーターによるメモリー・スキャンのオーバーヘッドの削減

lrubucket パラメーターをチューニングすると、大容量メモリー・システムのスキャン・オーバーヘッドを低減することができます。

ページ置換アルゴリズムは、フリー・フレームを探すためにメモリー・フレームをスキャンします。このスキャン中に、ページの参照ビットがリセットされ、フリー・フレームが見つからなかった場合は、2 回目のスキャンが行われます。 2 回目のスキャンでは、参照ビットがまだオフであれば、フレームは新しいページに使用されます (ページ置換)。

大容量メモリー・システムでは、スキャンするフレームが多すぎる場合があり、その場合、メモリーはフレームのバケットに分割されます。 ページ置換アルゴリズムでは、バケットの中のフレームをスキャンしてから、次のバケットへ移る前に、そのバケットの 2 回目のスキャンを開始します。 このバケット内のデフォルトのフレーム数は 131072、つまり 512 MB の RAM です。 フレームの数は、**vmo -o lrubucket**=*new value* コマンドでチューナブルで、またその値は 4 KB フレーム単位です。

minperm および maxperm パラメーターの値

オペレーティング・システムは、読み書きされたファイルをメモリー・ページに残しておくことによって、変化する実メモリーの要件を利用します。

ファイル・ページが再割り当てされる前に、そのファイル・ページが再び要求された場合、この手法では入出力操作を節約できます。 これらのファイル・ページは、ローカルまたはリモートの (例えば、NFS) ファイルシステムから使用できます。

ファイルに使用されるページ・フレームの、計算 (作業またはプログラム・テキスト) セグメントに使用されるページ・フレームに対する比率は、*minperm* 値と *maxperm* 値によって、緩やかに制御されています。

- ファイル・ページが占める RAM のパーセンテージが *minperm* より下がった場合は、ページ置換によりファイル・ページと計算ページの両方をスチールする。
- ファイル・ページが占める RAM のパーセンテージが *minperm* と *maxperm* の間にある場合は、ページ置換によりファイル・ページのみがスチールされる。

ある特定のワークロードでは、ファイル入出力を減らすことに重点を置く価値があり、別のワークロードにおいては、計算セグメント・ページをメモリーに保持することの方が重要な場合もあります。 未調整の状態で率がどのようになるかを理解するために、 **vmstat** コマンドを **-v** オプション付きで使用します。

```
# vmstat -v
              1048576 memory pages
               936784 lruable pages
```

```
683159 free pages
     1 memory pools
267588 pinned pages
  90.0 maxpin percentage
   3.0 minperm percentage
  90.0 maxperm percentage
   5.6 numperm percentage
 52533 file pages
   0.0 compressed percentage
     0 compressed pages
   5.6 numclient percentage
  90.0 maxclient percentage
 52533 client pages
     0 remote pageouts scheduled
     0 pending disk I/Os blocked with no pbuf
     0 paging space I/Os blocked with no psbuf
  2228 filesystem I/Os blocked with no fsbuf
    31 client filesystem I/Os blocked with no fsbuf
     0 external pager filesystem I/Os blocked with no fsbuf
  29.8 percentage of memory used for computational pages
```

numperm パーセンテージの値は、ファイル・セグメントが使用する実メモリーのパーセンテージを表します。値 5.6% は、メモリー内の 52533 ファイル・ページに相当します。

拡張 JFS ファイルシステム・キャッシュの制限 Maxclient

Maxclient は、バッファー・キャッシュに使用できるクライアント・ページの最大数を表します。

拡張 JFS ファイルシステムはクライアント・ページをバッファー・キャッシュとして使用します。 実メモリー内のクライアント・ページの制限は、*maxclient* を使用すると強制されます。

この LRU デーモンが実行開始する時点は、クライアント・ページ数が *maxclient* のしきい値の *minfree* ページ数の範囲内にある時です。この LRU デーモンがスチールしようとする対象クライアント・ページは、最近参照されていないページです。 ファイル・ページ数が *minperm* パラメーター値より小さい場合、参照されていないどのページも置換対象として選択される可能性があります。

Maxclient は、NFS クライアントと圧縮ページにも影響します。また、*maxclient* は、通常、*maxperm* パラメーターより小さいか等しい値に設定することにも注意してください。

ページ・スペース割り当て

AIX では、いくつかのページ・スペース割り当てポリシーが使用されます。

- 据え置きページ・スペース割り当て (DPSA)
- 早期ページ・スペース割り当て (EPSA)

据え置きページ・スペース割り当て

据え置きページ・スペース割り振りポリシーは、AIX のデフォルト・ポリシーです。

据え置きページ・スペース割り当てでは、ページング・スペースのディスク・ブロックの割り当ては、ページのページアウトが必要になるまで遅らせられ、その結果、ページ・スペースの割り当ての浪費がなくなります。 これにより、据え置きアルゴリズムは、使用できるページング・スペースよりも多くのページング・スペースを割り振ることができるようになります。その結果、ページング・スペースのオーバーコミットとなります。

ページがページング・スペースにページアウトされた後、そのページが RAM に再度ページインされると、そのページ用にディスク・ブロックが予約されます。 したがって、ページング・スペースの使用割合

の値は、その一部分が RAM にも戻されるので、必ずしもページング・スペース内のページ数だけを反映していない可能性があります。 ページ・バックされたページがスレッドの作業用ストレージである場合、そのスレッドがそのページと関連するメモリーを解放するかまたはそのスレッドが終了すると、そのページのディスク・ブロックは解放されます。 ディスク・ブロック (このブロックは、既に読み取られてメイン・メモリーに戻されているページ用のページング・スペース内にある) は、ページング・スペース・ガーベッジ・コレクション機能を使用して解放することができます。詳しくは、 171 ページの『ページング・スペース・ガーベッジ・コレクション』を参照してください。

ページング・スペース・ガーベッジ・コレクションが使用可能でない場合は、ページング・スペース量を適切に構成することが非常に重要です。 ファイル・キャッシュが minperm より少なく、十分なページング・スペースが構成されていない場合、ファイル・ページ・アクティビティーが原因で作業用ストレージのページがページアウトされないようシステムをチューニングすることが必要です。このワークロードで必要となる作業用ストレージ所要量が実メモリー量よりも少ない場合、およびファイル・ページ・アクティビティーが原因で作業用ストレージのページがページアウトされないようにシステムがチューニングされている場合、必要なページング・スペースは最小で済みます。据え置き割り振りセグメントではない、いくつかのページ・テーブル域 (PTA) セグメントは、内部 AIX カーネル・メモリー・セグメントと認識されます。これらのセグメントが必要とするページング・スペース予約を考慮するために、システムでは 512 MB のページング・スペースを推奨します。システムが大量の PTA スペースを使用すると、さらに多くのページング・スペースが必要になります。これは、**svmon -S** コマンドを使用して決定できます。

作業用ストレージ所要量が実メモリー量より多い場合は、作業用ストレージの仮想メモリー・サイズと少なくとも同量のページング・スペースを構成する必要があります。 そうしないと、最終的には、このシステムではページング・スペース不足となる可能性があります。

早期ページ・スペース割り当て

プロセスがページ不足のために kill されないようにしたい場合、このプロセスは、早期ページ・スペース割り振りポリシーを使用してページング・スペースを事前に割り当てることができます。

これは、*PSALLOC* と呼ばれる環境変数の値を early に設定することによって行います。 これはプロセスの中またはコマンド・ラインから (**PSALLOC=early** コマンド) 行います。 プロセスが **malloc()** サブルーチンを使用してメモリーを割り当てると、このメモリーにはこのプロセス用に予約されたページング・スペースのディスク・ブロックができます。すなわち、これらのディスク・ブロックがこのプロセス用に予約され、プロセスがページアウトを必要としたときには、常に使用可能なページング・スペースのスロットがあるという保証が与えられるわけです。 早期ページ・スペース割り振りポリシーを使用している場合、CPU の節約が重要なら、 **NODISCLAIM=true** と呼ばれる別の環境変数を設定して、 **free()** サブルーチン呼び出しのたびに、さらに **disclaim()** システム・コールにならないようにすることができます。

ページング・スペースおよび仮想メモリー

vmstat コマンド (*avm* 欄)、**ps** コマンド (*SIZE*、*SZ*)、およびその他のユーティリティーは、実際にアクセスされる仮想メモリーの量を報告します。DPSA では、ページング・スペースがタッチされない場合があるからです。

使用可能ページング・スペースを調べるには、**lsps -s** コマンドを使用した方が、**lsps -a** コマンドより安全です。**lsps -a** コマンドは、実際に使用されているページング・スペースを示すだけだからです。 しかし、**lsps -s** コマンドには、EPSA ポリシーで予約済みのページング・スペースと一緒に使おうとしているページング・スペースが含まれます。

ページング・スペースしきい値チューニング

使用可能なページング・スペースが使い尽くされて低レベルになった場合、オペレーティング・システムは、まずプロセスにページング・スペースを解放するように警告し、それでもまだ現在のプロセスに十分な使用可能スペースがない場合は、最後にプロセスを kill して、リソースを解放しようとします。

npswarn および npskill パラメーターの値

npswarn および *npskill* しきい値は、VMM によって、最初にプロセスに警告する時期、そして最後にプロセスを kill する時期を決定するために使用されます。

これら 2 つのパラメーターは、次に示す **vmo** コマンドによって設定できます。

npswarn

これは、オペレーティング・システムが SIGDANGER シグナルをプロセスへ送り始める、空きページング・スペースのページ数を指定します。 *npswarn* しきい値に達した場合、プロセスがこのシグナルを処理中であれば、プロセスは、それを無視するか、終了したり **disclaim()** サブルーチンを使用してメモリーを解放するなどの、何か他のアクションを行うかを選択することができます。

npswarn の値は、ゼロより大きく、システム上のページング・スペースの合計ページ数より小さくなければなりません。 これは **vmo -o npswarn=value** コマンドで変更できます。

npskill

これは、オペレーティング・システムがプロセスの kill を開始する、空きページング・スペースのページ数を指定します。 *npskill* しきい値に達すると、SIGKILL シグナルが一番若いプロセスに送られます。 SIGDANGER を処理中のプロセスまたは早期ページ・スペース割り当て (メモリーが要求されるとすぐにページング・スペースが割り当てられる) を使用しているプロセスは、kill を免除されます。 *npskill* のデフォルト値を決定する数式は、次のようになります。

```
npskill = maximum (64, number_of_paging_space_pages/128)
```

npskill の値は、ゼロより大きく、システム上のページング・スペースの合計ページ数より小さくなければなりません。 これは **vmo -o npskill=value** コマンドで変更できます。

nokilluid

vmo -o nokilluid コマンドで *nokilluid* オプションをゼロ以外の値に設定することによって、この値より低いユーザー ID は、低ページ・スペース条件のために kill されるのを免除されます。 例えば、**nokilluid** を 1 に設定すると、ルートが所有するプロセスは、**npskill** しきい値に達しても kill されるのを免除されます。

fork() 再試行間隔パラメーター

プロセスがページング・スペースのページ不足のために fork できない場合、スケジューラーは fork を 5 回再試行します。 各回の再試行の間に、スケジューラーはデフォルトで 10 クロックの目盛り分遅延します。

schedo コマンドの *pacefork* パラメーターは、失敗した **fork()** コールを再試行するまでに待つクロックの目盛りを指定します。例えば、**fork()** サブルーチン呼び出しが、新規プロセスの作成に十分な使用可能スペースがなかったために失敗した場合、システムは指定された数のクロックの目盛りだけ待ってから呼び出しを再試行します。 デフォルト値は 10 で、10 ms ごとに 1 クロックの目盛りがあるので、システムは **fork()** コールを 100 ms ごとに再試行します。

ワークロードのピークが短く、散発的であるということだけが理由でページング・スペースが少ない場合、再試行間隔を増やせば、以下のサンプルのようにプロセスが解放されるのに十分な時間遅延させることができます。

```
# schedo -o pacefork=15
```

この方法では、システムが **fork()** コールを再試行すると成功の可能性が高くなります。一部のプロセスがその間に実行を終了し、それにより、ページング・スペースのページが解放された可能性があるためです。

ページング・スペース・ガーベッジ・コレクション

一定の条件の下でページング・スペース・ガーベッジ・コレクション機能を使用してページング・スペースのディスク・ブロックを解放することができます。これによって、特定ワークロード用の仮想メモリー量と同じ量のページング・スペースを構成することは不要となります。 ガーベッジ・コレクション機能が使用可能なのは、据え置きページ・スペース割り振りポリシーに対してのみです。

再ページイン後のページング・スペース・ブロック上でのガーベッジ・コレクション

あるページがページング・スペースからメモリーに読み取られて戻された後でページング・スペースのディスク・ブロックを解放する方法は、デフォルトで使用されます。

これが再ページインごとに解放されない理由は、ブロックをページング・スペースに残しておくと、LRU デーモンがスチールした未変更の作業用ストレージのケースでは、パフォーマンスが向上するためです。ページがスチールされた場合、再ページアウト機能の実行は不要です。

vmo コマンドを使用して、以下のパラメーターをチューニングできます。

npsrpgmin パラメーターのチューニング:

項目	ディスクリプター
目的：	再ページイン・ガーベッジ・コレクションの開始時点での、空ページング・スペース・ブロック数のしきい値を指定します。
値：	デフォルト: MAX (768, **npswarn**+ (**npswarn**/2)
範囲：	0 から、システム内のページング・スペース・ブロック合計数まで

npsrpgax パラメーターのチューニング:

項目	ディスクリプター
目的：	再ページイン・ガーベッジ・コレクションの停止時点での、空ページング・スペース・ブロック数のしきい値を指定します。
値：	デフォルト: MAX (1024, **npswarn***2)

rpgclean パラメーターのチューニング:

項目	ディスクリプター
目的：	ページへの読み取りアクセス時に据え置きページ・スペース割り振りポリシーからのページのページング・スペース・ブロックを解放することを使用可能または使用不可にします。
値：	デフォルト: 0。この意味は、変更対象ページのページイン時のみ、ページング・スペースのディスク・ブロックを解放することを示します。 値 1 の意味は、変更またはアクセス対象ページ、つまり読み取り対象ページのページイン時にページング・スペースのディスク・ブロックを解放することを示します。
範囲：	0 ｜ 1

rpgcontrol パラメーターのチューニング:

項目	ディスクリプター
目的：	据え置きページ・スペース割り振りポリシーからのページのページイン時、ページング・スペース・ブロックの解放を使用可能または使用不可にします。
値：	デフォルト: 2。この意味は、ページイン時にページング・スペースのディスク・ブロックの解放が常に使用可能であることを示します。しきい値とは無関係です。 **注:** 読み取りアクセスが処理されるのは、 **rpgcontrol** パラメーター値が 1 の場合だけです。デフォルトでは、書き込みアクセスのみが常時処理されます。値 0 は、ページイン時にページング・スペースのディスク・ブロックの解放を使用不可にします。
範囲：	0 ｜ 1 ｜ 2

メモリーの消し込みによるガーベッジ・コレクション

ページング・スペース・ガーベッジ・コレクションの別の方法として、メモリーの消し込みがあります。この実装には、**psgc** カーネル・プロセスを使用します。

psgc カーネル・プロセスは、ページング・スペース・ディスク・ブロックを解放します。解放対象は、まだ再ページアウトされていない変更済みメモリー・ページか、または未変更のページ (そのページに対応したページング・スペース・ディスク・ブロックが存在する) です。

psgc カーネル・プロセスは、**vmo** コマンドを使用してチューニングできる、以下のチューニング・オプション・パラメーターを使用します。

npsscrubmin パラメーターのチューニングには以下のフィールドがあります。

項目	ディスクリプター
目的：	据え置きページ・スペース割り振りポリシーのページからのディスク・ブロックを解放するために、メモリー・ページの消し込みが開始する時点での、空いているページング・スペース・ブロックの数を指定します。
値：	デフォルト: MAX (768。**npsrpgmin** パラメーターの値)
範囲：	0 から、システム内のページング・スペース・ブロック合計数まで

npsscrubmax パラメーターのチューニングには以下のフィールドがあります。parameter includes the following fields:

項目	ディスクリプター
目的：	据え置きページ・スペース割り振りポリシーのページからのディスク・ブロックを解放するために、メモリー・ページの消し込みが停止する時点での、空いているページング・スペース・ブロックの数を指定します。
値：	デフォルト: MAX (1024。**npsrpgmax** パラメーターの値)
範囲：	0 から、システム内のページング・スペース・ブロック合計数まで

scrub パラメーターのチューニングには以下のフィールドがあります。

項目	ディスクリプター
目的：	メモリー内のページからページング・スペース用ディスク・ブロックを解放することを使用可能または使用不可にします。このディスク・ブロックは、据え置きページ・スペース割り振りポリシーからのものです。
値：	デフォルト: 0。メモリーの消し込みを完全に使用不可にします。 この値を 1 に設定すると、ページング・スペースのディスク・ブロックのメモリーの消し込みが使用可能になるのは次の時点です。すなわち、システムの空きページング・スペース・ブロック数が **npsscrubmin** パラメーター値より小さく、*npsscrubmax* パラメーター値より大きくなる時点です。
範囲：	0 ｜ 1

scrubclean パラメーターのチューニングには以下のフィールドがあります。

項目	ディスクリプター
目的：	メモリー内の未変更のページからページング・スペース用ディスク・ブロックを解放することを使用可能または使用不可にします。このディスク・ブロックは、据え置きページ・スペース割り振りポリシーからのものです。
値：	デフォルト: 0。メモリー内の変更済みページのみに対応するフリーのページング・スペースのディスク・ブロック数を示します。この値を 1 に設定すると、変更済みまたは未変更ページに対応するページング・スペースのディスク・ブロックを解放します。
範囲：	0	1

共有メモリー

shmat() または **mmap()** サブルーチンを使用すると、ファイルをメモリーに明示的にマップすることができます。 このプロセスにより、バッファリングとシステム・コールのオーバーヘッドを避けることができます。

メモリー領域は、共有メモリー・セグメントまたは領域として知られています。 以前に影響を受けた 32 ビット・アプリケーションの場合、共用ライブラリー・データまたは共用ライブラリー・テキスト・セグメントを含まない 11 個の共有メモリー・セグメントを提供するために、セグメント 14 がリリースされました。この方式は、セグメント 3 から 12 および 14 を使用したプロセスに適用されます。これらのセグメントは、それぞれ 256 MB のサイズです。 アプリケーションは、セグメントに対する読み取り/書き込みにより、ファイルの読み取り/書き込みを行うことができます。アプリケーションは、このようなマップされたセグメントのポインターを操作するだけで、読み取り/書き込みのシステム・コールのオーバーヘッドを避けることができます。

ファイルまたはデータは、複数のプロセスまたはスレッド間で共用することができます。ただし、これにはこれらのプロセス/スレッド間の同期が必要であり、そのような要求の処理はアプリケーションによって決まります。共有メモリーは通常、データベース・アプリケーションによって使用されます。この場合、データベースはラージ・データベース・バッファー・キャッシュとして使用されます。

ページング・スペースは、プロセスの専用セグメントの場合と同様、共有メモリー領域に割り当てられます。ページング・スペースは、据え置きページ・スペース割り当てポリシーがオフの場合、ページがアクセスされると使用されます。

拡張共有メモリー

拡張共有メモリーでは、32 ビット・プロセスに 1 バイトの共有メモリー・セグメント (ただし、これは最も近いページに切り上げられます) を割り当てることができます。 このフィーチャーは、プロセス環境で変数 **EXTSHM** が **ON**、**1SEG**、または **MSEG** のいずれかに設定されているプロセスだけが使用できます。

拡張共有メモリーについては、基本的に 11 個のみの共有メモリー領域の制限は排除されます。 64 ビット・プロセスは **EXTSHM** 変数の影響を受けません。

EXTSHM を **ON** に設定すると、この変数を **1SEG** に設定した場合と同じ効果があります。 いずれの設定でも 256 MB より小さい共有メモリーが mmap セグメントとして内部に作成され、パフォーマンスへの影響は mmap と同じです。 作業セグメントには 256 MB 以上の共有メモリーが内部に作成されます。

EXTSHM を **MSEG** に設定すると、すべての共有メモリーは mmap セグメントとして内部に作成され、メモリーの使用率を向上させます。

プロセスが接続できる共有メモリー領域の数には制限がありません。 ファイル・マッピングは前と同様にサポートされますが、 256 MB (セグメント・サイズ) の倍数のアドレス・スペースを消費します。 共有メモリー領域のサイズ変更は、このモードではサポートされません。 カーネル・プロセスは同じ動作をします。

拡張共有メモリーには以下の制約事項があります。

- 入出力サポートはメモリー・マップ領域と同じように制限される。
- **uphysio()** タイプの入出力だけがサポートされる (ロー I/O はサポートされない)。
- これらの共有メモリー領域は、割り込みハンドラーでバッファーのピン取り消しが起こる入出力バッファーとして使用することはできない。 例えば、これらの領域は非同期通信 I/O バッファーには使用できません。
- セグメントは **plock()** サブルーチンを使用して固定することはできない。メモリー・マップされたセグメントは **plock()** サブルーチンで固定できないからです。

1 TB セグメントの別名割り当て

1 TB セグメントのエイリアス・アドレスは、256 MB セグメント・サイズの共有メモリー領域上で 1 TB セグメント変換を使用することにより、パフォーマンスを向上させます。 このサポートは、共有メモリー領域を使用する、すべての 64 ビット・アプリケーションで提供されています。 明示的および非明示的な共有メモリー接続 (shmat) について、どちらの場合でも、1 TB セグメントのエイリアス・アドレスを使用できます。

アプリケーションがその共有メモリー領域で 1 TB エイリアスを使用する資格がある場合、AIX オペレーティング・システムはアプリケーションを変更することなく 1 TB セグメント変換を使用します。 この場合は、`shm_1tb_shared VMO` チューナブル、`shm_1tb_unshared VMO` チューナブル、および `esid_allocator VMO` チューナブルを使用する必要があります。

`shm_1tb_shared VMO` チューナブルは、*"SHM_1TB_SHARED=" VMM_CNTRL* 環境変数を使用してプロセスごとに 1 TB 共有メモリーを設定することができます。 デフォルト値は、プロセッサーの機能に応じてブート時に動的に設定されます。 1 つの共有メモリー領域に、ここで要求される数の ESID が存在すれば、自動的にエイリアスの共有メモリー領域に変更されます。 許容値は 0 から 4 KB の範囲内です (1 TB あたりおおよそ 256 MB の ESID を必要とします)。

`shm_1tb_unshared VMO` チューナブルは、*"SHM_1TB_UNSHARED=" VMM_CNTRL* 環境変数を使用してプロセスごとに 1 TB 非共有メモリーを設定することができます。 デフォルト値は 256 に設定されています。 許容値は 0 から 4 KB の範囲内です。 このデフォルト値は慎重に設定されており (最大 64 GB までアドレス・スペースの割り当てを要求します)、それを超えると、非共用の 1 TB エイリアスに移行します。 このしきい値が 256 MB セグメントに設定され、これを超えると共有メモリー領域で 1 TB エイリアスを使用するようにプロモートされます。 これより低い値は慎重に検討した上で、共有メモリー領域を 1 TB エイリアスで使用する必要があります。これにより、セグメント・ルックアサイド・バッファー (SLB) ミスが減る可能性がありますが、多くの共有メモリー領域をプロセス間にまたがって使用せずに、エイリアス割り当てにする場合は、ページ・テーブル・エントリー (PTE) ミスが増える可能性もあります。

`esid_allocator VMO` チューナブルは、*"ESID_ALLOCATOR=" VMM_CNTRL* 環境変数を使用してプロセスごとに ESID アロケーターを設定することができます。 デフォルト値は、AIX バージョン 6.1 では 0、AIX バージョン 7.0 では 1 に設定されています。 0 または 1 のいずれかの値を指定できます。0 に設定した場合は、非明示的な共有メモリー接続には古いアロケーターが有効になります。 それ以外の場合は、非明示的な接続に対して新しいアドレス・スペース割り当てポリシーが使用されます。 この新しいアドレス・スペース・アロケーターは、(SHM や MMAP などの) 非明示的な割り当てをアプリケーションのアドレス・スペースの新しいアドレス範囲 0x0A00000000000000 - 0x0AFFFFFFFFFFFFFF に接続しま

す。 アロケーターは、1 TB エイリアスにプロモーションするチャンスを最大限に利用できるように割り当てを最適化します。 このような最適化により、アドレス・スペースの「穴」が生じることがあります。これは非明示的な接続を使用している場合は正常と見なされます。明示的接続は、0x0700000000000000 - 0x07FFFFFFFFFFFFFF の範囲に行われ、旧バージョンとの互換性が保持されます。 この新しい割り当てポリシーによりバイナリーの互換性問題が発生する特定の状況では、このチューナブルを 0 に設定することで既存のアロケーター動作を復元することができます。

共有エイリアスへのプロモーションの対象とならなかった複数の共有メモリー領域は、1 TB 領域としてグループ化されます。 アプリケーションのアドレス・スペースの 1 TB 領域にある共有メモリー領域のグループは、アプリケーションが 256 MB セグメントのしきい値を越えると、非共有の 1 TB エイリアスを使用するようにプロモートされます。 共有メモリーの接続と切り離しが頻繁に行われるアプリケーションでは、共有されないエイリアスしきい値の値を低くすると、パフォーマンスが低下します。

環境変数の名前空間を汚染しないように、すべての環境チューナブルがマスター・チューナブル `VMM_CNTRL` の下で使用されます。このマスター・チューナブルは、`@` 記号を使用してコマンドを区切って指定されます。 `VMM_CNTRL` の使用例を次に示します。

```
VMM_CNTRL=SHM_1TB_UNSHARED=32@SHM_1TB_SHARED=5
```

すべての環境変数設定が `fork()` で子に継承され、`exec()` でシステム・デフォルト値に初期化されます。32 ビット・アプリケーションはすべて VMO または環境変数チューナブルの変更による影響を受けません。

すべての VMO チューナブルと環境変数について、**vm_pattr** に類似したコマンドがあります。ただし、`esid_allocator` チューナブルは例外です。 コマンドを実行する前に共有メモリー・アドレス・スペースの一部が割り当てられる状況を回避するために、このチューナブルは `vm_pattr` オプションには存在していません。

AIX メモリー親和性サポート

AIX では、ページ・フォールトの原因となったプロセッサーが含まれているモジュールからメモリーをプロセス用に割り当てることができます。 この能力は、システム上で、*MEMORY_AFFINITY* 環境変数を設定することによってメモリー親和性サポートが使用可能になっている場合に使用できます。 メモリー親和性はデフォルトで使用可能ですが、使用不可にすることができます。

IBM POWER プロセッサー・ベースのプラットフォーム SMP ハードウェア・システムには、個別のシステムに応じて、シングル、デュアル、または複数のプロセッサー・チップのサポートに対応したモジュールが含まれています。 各モジュールには複数のプロセッサーが含まれており、これらのモジュールにシステム・メモリーが接続されています。 プロセッサーはシステム内のすべてのメモリーにアクセスできますが、プロセッサー自身のモジュールに接続されたメモリーをアドレッシングするときは、システム内の他のモジュールに接続されたメモリーをアドレッシングするときより高速にアクセスでき、より高い帯域幅を獲得できます。

メモリー親和性が使用可能であると、各モジュールは独自の vmpool を持ちます。これには、1 つ以上のメモリー・プール数が含まれます。 各メモリー・プールには、独自のページ置換デーモン **lrud** があります。 各プール内のメモリー量は、モジュール内で使用可能なメモリー量またはハイパーバイザー・レイヤーによって VMM に割り当てられたメモリー量に基づきます。

AIX オペレーティング・システムを使用しており、メモリー親和性が使用不可になっている場合、メモリー・プール数は、メモリー量およびシステム内の CPU 数に基づきます。

AIX 上でメモリー親和性サポートを使用不可にするには、以下の **vmo** コマンドを使用します。

```
vmo -o memory_affinity=0
```

注**:** このコマンドを有効にするには、`bosboot` と `reboot` が必要です。
デフォルト値は 1 で、これは、メモリー親和性サポートが使用可能であることを意味します。

メモリー親和性サポートを使用可能にすると、オペレーティング・システムに対して、システムのデータ構造をモジュール境界に合わせて編成するよう指示されます。 デフォルト・メモリー割り振りポリシーが、MCM 全体に回覧されます。 アプリケーションがローカル MCM のメモリー割り当てを優先的に取得するには、次のように MEMORY_AFFINITY 環境変数をエクスポートする必要があります。

```
MEMORY_AFFINITY=MCM
```

この動作は、fork 全体に伝搬されます。 ただし、**exec** 関数への呼び出し以降もこの動作を保持するには、この変数を、**exec** 関数呼び出しに渡された環境ストリングに含める必要があります。

関連情報

vmo コマンドおよび 164 ページの『VMM ページ置換のチューニング』

bindprocessor コマンドまたはサブルーチン

WLM クラス属性およびリソース・セット属性

ローカル MCM メモリー割り当てによるパフォーマンスへの影響

ローカル MCM メモリー割り当てによる特定のアプリケーションへの影響は、予測が困難です。 アプリケーションの中には影響を受けないものがあり、パフォーマンスが向上するものも、低下するものもあります。

ほとんどのアプリケーションは、メモリー親和性によってパフォーマンスを向上させるために、プロセッサーにバインドする必要があります。 アプリケーションの実行中に、AIX ディスパッチャーがそのアプリケーションを別の MCM のプロセッサーに移動させないために、この操作が必要です。

メモリー親和性によってパフォーマンスを向上させるための最適な方法は、アプリケーションの実行を、1 つの MCM に含まれるプロセッサーだけに制限することです。この操作は、**bindprocessor** コマンドと **bindprocessor()** 関数を使用して実行できます。 また、**resource set** 親和性コマンドおよびサービスでも実行できます。

1 つの MCM に含まれているプロセッサーより多くのプロセッサーをアプリケーションが必要とする場合、メモリー親和性によってパフォーマンスが向上するかどうかは、そのアプリケーションのさまざまなスレッドのメモリー割り当てと、アクセス・パターンによって決まります。アプリケーションのスレッドが、それぞれ個別に固有のデータ域を割り当て、参照する場合、パフォーマンスが向上する可能性があります。すべてのスレッドの間でメモリーを共用するアプリケーションは、メモリー親和性によってパフォーマンスが低下する可能性が最も高くなります。

vmo コマンドによるメモリー配置

vmo コマンドのパラメーターを使用してユーザー・メモリーを割り当てることができます。ファースト・タッチ・スケジューリング・ポリシーを使用するのか、または周期的スケジューリング・ポリシーを使用するのかを決定することもできます。

ファースト・タッチ・スケジューリング・ポリシーの場合、メモリーは、最初にそのメモリー・セグメントにタッチしたとき (最初のページ・フォールト) にスレッドが実行していたチップ・モジュールから割り当

てられます。 周期的スケジューリング・ポリシーの場合、すべてのメモリー・タイプのデフォルトですが、メモリー割り当ては、各 vmpools についてストライプされます。

以下の **vmo** コマンドのパラメーターには、ユーザー・メモリーの配置を制御し、ファースト・タッチ・スケジューリング・ポリシーを示す値 1、または周期的スケジューリング・ポリシーを示す値 2 のどちらかを指定できます。

memplace_data
このパラメーターは、以下のタイプのデータのためのメモリー配置を指定します。
- 初期化または未初期化のメイン実行可能モジュールのデータ
- ヒープ・セグメント・データ
- 共用ライブラリー・データ
- 実行時にロードされたオブジェクト・モジュールのデータ

このパラメーターのデフォルト値は 2 です。

memplace_mapped_file
このパラメーターはプロセス (**shmat()** 関数および **mmap()** 関数など) のアドレス・スペースにマップされるファイル用のメモリー配置を指定します。 このパラメーターのデフォルト値は 2 です。

memplace_shm_anonymous
このパラメーターは、**shmget()** 関数または **mmap()** 関数への呼び出しによって作成される作業記憶域として機能する無名共有メモリーのためのメモリー配置を指定します。 このメモリーは、プロセスまたはその子孫を作成することによってのみアクセス可能であり、名前またはキーと関連付けられません。このパラメーターのデフォルト値は 2 です。

memplace_shm_named
このパラメーターは、**shmget()** 関数または **shm_open()** 関数への呼び出しによって作成される作業記憶域として機能する名前付き共有メモリーのためのメモリー配置を指定します。 これは、複数のプロセスが同時にアクセスできるようにする名前またはキーと関連付けられます。 このパラメーターのデフォルト値は 2 です。

memplace_stack
このパラメーターは、プログラム・スタックのためのメモリー配置を指定します。 このパラメーターのデフォルト値は 2 です。

memplace_text
このパラメーターは、メイン実行可能モジュールのアプリケーション・テキスト用のメモリー配置を指定し、その依存関係用のメモリー配置ではありません。 このパラメーターのデフォルト値は 2 です。

memplace_unmapped_file
このパラメーターは、**read()** または **write()** 関数などの場合、マップ解除されたファイル・アクセス用のメモリー配置を指定します。 このパラメーターのデフォルト値は 2 です。

MEMORY_AFFINITY 環境変数によるメモリー配置

このプロセス・レベルでは、*MEMORY_AFFINITY* 環境変数を使用してユーザー・メモリーの配置を構成できます。これは、**vmo** コマンドのパラメーターを指定してメモリー配置をオーバーライドします。

以下の表に、*MEMORY_AFFINITY* 環境変数の可能な値をリストします。

値	振る舞い
MCM	専用メモリーがローカルで、共有メモリーがローカルです。
SHM=RR	System V と Posix Real-Time の両方の共有メモリーが MCM 全体にわたってストライプされます。 4 KB のラージ・ページ対応共有メモリー・オブジェクトに適用されます。 この値は、64 ビット・カーネルの場合のもので、MCM 値も定義されている場合にのみ有効です。
LRU=EARLY	LRU デーモンは、下限しきい値 (*minfree* パラメーターなど) に達するとすぐに、ローカル・メモリーで開始します。 すべてのシステム・プールが下限しきい値に達するまで待機しません。 この値は、MCM 値も定義されている場合にのみ有効です。

アットマーク (@) で各値を区切ると、*MEMORY_AFFINITY* 環境変数に複数の値を設定できます。

ラージ・ページ

ラージ・ページを使用する主な目的は、大量の仮想メモリーを使用するハイパフォーマンス・コンピューティング (HPC) アプリケーションまたはメモリー・アクセス集中型アプリケーションのシステム・パフォーマンスを向上させることです。 システム・パフォーマンスが向上する原因は、変換索引バッファー (TLB) がマップできる仮想メモリーの範囲が広がるので、TLB のミスが減ることです。

ラージ・ページでは、4 KB の境界での事前取り出し操作の再始動の必要性をなくすために、メモリーの事前取り出しも改善されています。 AIX は 32 ビットと 64 ビット両方のアプリケーションによるラージ・ページの使用をサポートしています。

POWER4 ラージ・ページ・アーキテクチャーでは、256 MB セグメント内のすべての仮想ページが同サイズである必要があります。 AIX は、プロセス内のいくつかのセグメントは 4 KB ページで対応し、他のセグメントは 16 MB ページで対応するという混合モード処理モデルを使用することによって、このアーキテクチャーをサポートしています。アプリケーションは、ヒープ・セグメントまたはメモリー・セグメントをラージ・ページで対応するように要求することができます。 詳しくは、『ラージ・ページを使用するためのアプリケーション構成』を参照してください。

AIX は 4 KB と 16 MB の別々の物理メモリー・プールを維持します。 16 MB メモリー・プールでの物理メモリー量を、**vmo** コマンドを使用して指定することができます。 ラージ・ページ・プールは動的になったため、指定した物理メモリー量は即時に有効になり、システム・リブートの必要はありません。残りの物理メモリーは 4 KB 仮想ページに対応します。

AIX はラージ・メモリーを固定メモリーとして扱います。 AIX は、ラージ・ページでのページングはサポートしていません。 ラージ・ページで対応するアプリケーションのデータは、アプリケーションが完了するまで物理メモリーに残ります。セキュリティー・アクセス制御メカニズムにより、無許可アプリケーションはラージ・ページまたはラージ・ページ物理メモリーを使用できないようになっています。 また、セキュリティー・アクセス制御メカニズムは、無許可ユーザーによるアプリケーションでのラージ・ページの使用を防止します。 root 以外のユーザー ID でラージ・ページを使用するには、**chuser** コマンドで **CAP_BYPASS_RAC_VMM** 能力を使用可能にする必要があります。 以下は、**CAP_BYPASS_RAC_VMM** 能力としてスーパーユーザーを付与する方法の例です。

```
# chuser capabilities=CAP_BYPASS_RAC_VMM,CAP_PROPAGATE <user id>
```

ラージ・ページを使用するためのアプリケーション構成

ラージ・ページを使用するようにアプリケーション構成するには、幾つかの方法があります。

データおよびヒープ・セグメントに対応するラージ・ページの使用**:**

アプリケーションでのラージ・ページ・データまたはヒープの使用は、そのアプリケーションを実行するときに決定する必要があります。これは、アプリケーションは、実行開始後はモードを切り替えることができないためです。 ラージ・ページの使用は、**fork()** 関数の子プロセスによって継承されます。

以下の方法でアプリケーションを構成して、初期化されたプログラム・データ、未初期化プログラム・データ (BSS)、およびヒープ・セグメントのラージ・ページ対応を要求することができます。

- 『ラージ・ページを要求するための実行可能ファイルへのマーキング』
- 『ラージ・ページを要求するための環境変数の設定』

アプリケーションがデータまたはヒープ・セグメントにラージ・ページを使用するようにするときのモードとして、以下のいずれかを指定できます。

- 180 ページの『状況依存モード』
- 180 ページの『必須モード』

データおよびヒープ・セグメントにラージ・ページを使用する 32 ビット・アプリケーションは、ラージ・ページのページ保護細分度のために、ラージ・ページ 32 ビット処理モデルを使用します。 他の処理モデルでは、同一セグメント内でさまざまな保護属性を持つ 4 KB ページが使用されますが、これは保護細分度が 16 MB のときは機能しません。

ラージ・ページを要求するための実行可能ファイルへのマーキング*:*

実行可能ファイルの XCOFF ヘッダーに **blpdata** フラグがあります。このフラグは、アプリケーションがデータおよびヒープ・セグメントに対応するラージ・ページの使用を望んでいることを示します。

実行可能ファイルへのラージ・ページ要求のマーキングは、次のコマンドを使用します。

```
# ldedit -blpdata <filename>
```

データおよびヒープ・セグメントに対応するラージ・ページを使用しないことにする場合は、次のコマンドを使用してラージ・ページ・フラグをクリアします。

```
# ldedit -bnolpdata <filename>
```

cc コマンドでリンクおよびバインド時、**blpdata** オプションも設定できます。

ラージ・ページを要求するための環境変数の設定*:*

LDR_CNTRL 環境変数を使用して、アプリケーションのデータおよびヒープ・セグメントにラージ・ページを使用するようにアプリケーションを構成することができます。 この環境変数は、実行可能ファイルの **blpdata** フラグより優先します。

LDR_CNTRL 環境変数では、以下のオプションを使用できます。

- **LDR_CNTRL=LARGE_PAGE_DATA=Y** オプションは、実行されるアプリケーションがそのデータおよびヒープ・セグメントにラージ・ページを使用することを指定します。このオプションは、実行可能ファイルへのラージ・ページ使用のマーキングと同じ意味です。
- **LDR_CNTRL=LARGE_PAGE_DATA=N** オプションは、実行されるアプリケーションがそのデータおよびヒープ・セグメントにラージ・ページを使用しないことを指定します。このオプションは、実行可能ファイルへのラージ・ページ使用のマーキング設定をオーバーライドします。
- **LDR_CNTRL=LARGE_PAGE_DATA=M** オプションは、実行されるアプリケーションがそのデータおよびヒープ・セグメントにラージ・ページを必須モードで使用することを指定します。

注**:** ラージ・ページ環境変数の設定は、ラージ・ページを使用することで利点が得られると考えられる特定のアプリケーションについてのみ行ってください。 そうしないと、システムに何らかのパフォーマンス低下が起きる可能性があります。

状況依存モード*:*

状況依存モードでは、アプリケーションのヒープ・セグメントの一部はラージ・ページで対応し、他は 4 KB ページで対応する可能性があります。 セグメントに対応できるだけのラージ・ページがないときは、データまたはヒープ・セグメントの対応に 4 KB ページが使用されます。

状況依存モードでは、アプリケーションは、以下の前提条件が可能な場合にラージ・ページを使用します。

- ユーザー ID にラージ・ページの使用が許可されている。
- システム・ハードウェアにラージ・ページ・アーキテクチャー機構がある。
- ラージ・ページ・メモリー・プールを定義済みである。
- ラージ・ページ・メモリー・プールに、ラージ・ページを使用するセグメント全体に対応できるだけのページがある。

これらの条件の 1 つでも満たされないと、アプリケーションのデータおよびヒープ・セグメントは 4 KB ページでの対応になります。

ラージ・ページを使用するとしてマークされた実行可能ファイルは、状況依存モードで実行されます。

必須モード*:*

必須モードでは、アプリケーションがヒープ・セグメントを要求してもその要求に対応できるだけのラージ・ページがない場合は、そのアプリケーション要求は失敗します。この場合、ほとんどのアプリケーションはエラーで終了します。

必須モードを使用する場合は、ラージ・ページ・プールのサイズをモニターして、プールのラージ・ページが使い尽くされていないことを確認してください。そうしないと、必須モードのラージ・ページ・アプリケーションは失敗します。

共有メモリー・セグメントに対応するラージ・ページの使用**:**

アプリケーションの共有メモリー・セグメントをラージ・ページで対応させるには、**shmget()** 関数の **SHM_LGPAGE** および **SHM_PIN** フラグを指定します。ラージ・ページを使用できない場合は、4 KB ページが共有メモリー・セグメントの対応に使用されます。

ラージ・ページ共有メモリーと、ラージ・ページ・データおよびヒープ・セグメントに対応する物理メモリーは、ラージ・ページ物理メモリー・プールのものが使用されます。 共有メモリー・ラージ・ページとデータおよびヒープ・ラージ・ページを使用できるだけのラージ・ページが、ラージ・ページ物理メモリー・プールにあることを確認する必要があります。

ラージ・ページを使用するためのシステム構成

ラージ・ページを使用するには、システムを構成し、ラージ・ページ対応に割り当てる物理メモリー量も指定する必要があります。

デフォルトでは、システムにはラージ・ページ物理メモリー・プールに割り当てられたメモリーはありません。 **vmo** コマンドを使用して、ラージ・ページの物理メモリー・プールのサイズを構成できます。 次の例では、ラージ・ページの物理メモリー・プールに 1 GB のメモリーを割り当てます。

```
# vmo -r -o lgpg_regions=64 -o lgpg_size=16777216
```

共有メモリーにラージ・ページを使用するには、次のコマンドで **SHM_PIN shmget()** システム・コールを使用可能に設定する必要があります。この設定はシステム・リブートをしても持続します。

```
# vmo -p -o v_pinshm=1
```

システムで使用されているラージ・ページの数を調べるには、**vmstat -l** コマンドを使用します。以下に例を示します。

```
# vmstat -l

kthr     memory             page              faults        cpu      large-page

----- ----------- ------------------------ ------------ ----------- ------------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa   alp   flp
 2  1 52238 124523   0   0   0   0    0   0 142   41  73  0  3 97  0     16     16
```

この例から、16 のアクティブ・ラージ・ページ (alp) と 16 のフリー・ラージ・ページ (flp) があることが分かります。

ラージ・ページを使用する場合の考慮事項

ラージ・ページのサポートは、特殊目的のパフォーマンス改善用フィーチャーなので、一般に使用することはお勧めできません。 すべてのアプリケーションがラージ・ページを使用することで利点が得られるとは限りません。 実際、多数の **fork()** 関数を実行するアプリケーションなど、アプリケーションによっては、ラージ・ページを使用するとパフォーマンス低下を起こしやすいものもあります。

ラージ・ページを使用する場合は、*LDR_CNTRL* 環境変数を使用するのではなく、特定の実行可能ファイルへのマーキングを検討してください。これは、ラージ・ページを使用することで利点が得られる特定のアプリケーションにラージ・ページの使用を限定するためです。

ラージ・ページの使用を検討する場合は、システムの全体パフォーマンスへの影響を考えてください。 ラージ・ページを使用することでいくつかの特定のアプリケーションでは利点が得られても、システム全体のパフォーマンスから見ると、システムで使用可能な 4 KB ページ・ストレージが減ったために、パフォーマンス低下が見られる場合があります。 4 KB ページの数が減ってもシステムのパフォーマンスの著しい妨げにならないような十分な物理メモリーがシステムにある場合に、ラージ・ページの使用を検討してください。

複数ページ・サイズのサポート

POWER5+ プロセッサーは 4 種類の仮想メモリー・ページ・サイズ (4 KB、64 KB、16 MB、および 16 GB) をサポートします。 また、IBM Power Systems プロセッサー・ベースのサーバーは、ベース・ページ・サイズ 4KB のセグメントでの 64 KB ページの使用もサポートします。 AIX はこのプロセスを使用して、有用な場合に 64 KB ページのパフォーマンス上の利点を提供したり、64 KB ページだと多くのメモリーが無駄になる場合 (割り当てられてはいるが、アプリケーションによって使用されないなど) に 4 KB ページを用います。

アプリケーションのメモリーとして 64 KB のようなより大きい仮想メモリー・ページ・サイズを使用すると、アプリケーションのパフォーマンスとスループットは、ページ・サイズの大きさに見合うハードウェア効率により、著しく改善されます。 大きなページ・サイズを使用すると、仮想ページ・アドレスから物理ページ・アドレスへの変換のためのハードウェア待ち時間を減少させることができます。 この待ち時間の減少は、プロセッサーの Translation Lookaside Buffer (TLB) と同じようにハードウェア変換キャッシュ効率の改善によるものです。 ハードウェア変換キャッシュはエントリー数だけが制限されているために、より大きいページ・サイズを使用することで、キャッシュにあるエントリー当たりの変換できる仮想メモリー量が増えるからです。これにより、アプリケーションでアクセスできるメモリー量を増やすことが可能であり、ハードウェア変換により遅延されることもありません。

16 MB および 16 GB ページは超ハイパフォーマンス環境のみを対象にしていますが、64 KB ページは汎用であり、たいていのワークロードについては 4 KB ページより 64 KB ページを使用するほうが明らかに優れています。

プロセッサー・タイプごとのサポートされるページ・サイズ

特定システムでは、AIX でサポートされるすべてのページ・サイズを判別するには、**-a** オプションを指定した **pagesize** コマンドを使用します。

AIX 6.1 以降は、4 KB と 64 KB の 2 つのページ・サイズのセグメントをサポートします。デフォルトで、プロセスはこれらの可変ページ・サイズ・セグメントを使用します。 これは既存のページ・サイズ選択メカニズムによってオーバーライドされます。

表 2. *AIX* および各種 *System p* ハードウェアでサポートされるページ・サイズ

ページ・サイズ	必要なハードウェア	ユーザー構成の必要	制限
4 KB	ALL	なし	なし
64 KB	POWER5+ 以降	なし	なし
16 MB	POWER4 以降	あり	あり
16 GB	POWER5+ 以降	あり	あり

表 3. サポートされるセグメント・ページ・サイズ

セグメント・ベース・ページ・サイズ	サポートされるページ・サイズ	最小限必要なハードウェア
4 KB	4 KB/64 KB	POWER6®
64 KB	64 KB	POWER5+
16 MB	16 MB	POWER4
16 GB	16 GB	POWER5+

前のバージョンの AIX と同様に、4 KB はデフォルト・ページ・サイズです。ユーザーが別のページ・サイズの使用を要求しなければ、プロセスは 4 KB ページを使用して継続されます。

64 KB ページ・サイズのサポート

64 KB ページ・サイズは使用が簡単であることと、多くのアプリケーションで 4 KB ページ・サイズよりも 64 KB ページ・サイズを使用するほうが優れたパフォーマンスを期待できるため、AIX では 64 KB ページ・サイズのサポートは豊富です。

64 KB ページ・サイズを使用可能にするために、システムにはどのようなシステム構成の変更も必要ありません。 64 KB ページ・サイズをサポートするシステムでは、AIX カーネルが使用するシステムを自動構成します。 サイズが 64 KB のページは完全にページング可能であり、そして、システムの 64 KB ページ・フレームのプール・サイズは動的であり、AIX により完全に管理されます。 AIX は異なるページ・サイズ要求を満たすために、システムの 4 KB ページ・フレームと 64 KB ページ・フレームの数を変えます。 **svmon** コマンドおよび **vmstat** コマンドはいずれも、システムの 4 KB および 64 KB ページ・フレームの数をモニターする場合に使用できます。

動的可変ページ・サイズのサポート

POWER6 より前のプロセッサーでは、セグメントごとに単一のページ・サイズのみがサポートされていました。 システム管理者またはユーザーは、メモリー占有スペースを基に、特定のアプリケーションに最適のページ・サイズを選択しなければなりませんでした。

4 KB のページを選択した場合、実際に参照されたこれらの 4 KB のページのみが使用されたため、無駄になるメモリー量は最小でした。 ページ・サイズが大きくなると、実効ページ・セットの場所によっては (割り振られてはいるが、使用されないといったことで) 多くのメモリーが無駄になる可能性があり、明らかなパフォーマンス改善には仮想から物理への変換をより少なくする必要が生じることがあります。 さらに、4 KB を超えるページ・サイズには、明示的に特定のページ・サイズを選択するためのユーザー介入が必要です。

POWER6 では、単一セグメント内での混合ページ・サイズの概念が導入されました。 このアーキテクチャーは、異なるページ・サイズのさまざまな置換をサポートします。ただし、POWER6 では 4 KB と 64 KB のページ・サイズの混合のみがサポートされます。 AIX 6.1 はこの新しいハードウェア能力を利用して、参照回数がわずかなメモリー領域での 4 KB ページ・サイズの保守的なメモリー使用率の側面を、参照回数が多いメモリー領域での 64 KB ページ・サイズのパフォーマンス上の利点を組み合わせます。そしてこれはユーザー介入なしに自動的に行われます。 この AIX フィーチャーは、Dynamic Variable Page Size Support (VPSS) と呼ばれます。 後方互換性の問題を避けるために、現在ユーザーによって明示的ページ・サイズが選択されているセグメントでは VPSS は使用不可になっています (『複数ページ・サイズ・アプリケーション・サポート』を参照)。

可変ページ・サイズ・セグメントのデフォルト設定は、16 すべての 4KB のページが参照されるまで、64 KB サイズで位置合わせされた領域で 4KB ページと 4KB の変換を使用します。 16 ページすべてが参照されると、そのすべてが同じ状態であることを確認するための検査が行われます (同一のページ読み取り/書き込みの保護、no-exec 保護、ストレージ・キー保護、および入出力状態になっていないことなど)。 確認が済むと、4 KB 変換が除去されて、64 KB 変換に置き換えられます。

64 KB 変換は、すべての 16 KB ページが継続して同一状態であるかぎり使用されます。 そのうちの 1 つのページの状態が (例えば、**mprotect** サブルーチンを通じて、あるいは LRU がそのうちの 1 つをスチーリングしたことにより) 変更されると、同一状態を回復するまで通常の 4 KB ページにデモートされます。

アプリケーションの中には、64 KB 領域が完全に参照されない場合でも、より大きなページ・サイズの使用を好むものがあります。 Page Size Promotion Aggressiveness Factor (PSPA) を使用すると、4 KB ページのグループが 64 KB ページ・サイズにプロモートされるメモリー参照要件を減らすことができます。PSPA は、**vmm_default_pspa vmo** チューナブルを使用してシステム全体に対して設定するか、**vm_pattr** コマンドを使用して特定のプロセスに対して設定することができます。

64 KB ページ・サイズのサポートと同様に、**svmon** コマンドは可変ページ・サイズの使用率を報告するように更新されています。 **vmo** コマンドについて詳しくは、*Commands Reference, Volume 6*を参照してください。

超ハイパフォーマンス環境のためのページ・サイズ

4 KB および 64 KB ページ・サイズに加えて、AIX は 16 MB ページ (ラージ・ページ とも呼ばれます) および 16 GB (ヒュージ・ページ・サイズ とも呼ばれます) をサポートしています。 上記のページ・サイズはハイパフォーマンス環境のみで使用されることを目的としています。したがって、AIX では、これらのページ・サイズを使用するシステムは自動構成されません。

これらのページ・サイズを使用するには、AIX を構成する必要があります。 ページ・サイズごとにページ数も構成する必要があります。 AIX は、構成済みの 16 MB または 16 GB ページ数を要求時対応ベースで自動変更できません。

16 MB ラージ・ページに割り当てられたメモリーは 16 MB ラージ・ページのためだけに使用可能であり、16 GB ヒュージ・ページに割り当てられたメモリーは 16 GB ヒュージ・ページのためだけに使用可

能です。 したがって、これらのサイズのページはハイパフォーマンス環境だけで構成してください。 さらに、16 MB および 16 GB ページの使用には、このページ・サイズのページを割り当てるために、「ユーザーは **CAP_BYPASS_RAC_VMM** および **CAP_PROPAGATE** 機能、または **root** 権限を持っている必要がある」という制限があります。

ラージ・ページ数の構成**:**

16 MB ラージ・ページ数をシステムに構成するには、**vmo** コマンドを使用します。

16 MB ラージ・ページを 1 GB 割り当てる例を示します。

```
# vmo -r -o lgpg_regions=64 -o lgpg_size=16777216
```

上の例のラージ・ページの構成変更は、**bosboot** コマンドを実行し、システムをリブートするまで有効になりません。 動的ロジカル・パーティショニング (DLPAR) をサポートするシステムでは、上に示したコマンドから **-r** オプションを省略すると、システムをリブートしないで 16 MB ラージ・ページを動的に構成できます。

16 MB ラージ・ページについて詳しくは、 178 ページの『ラージ・ページ』を参照してください。

ヒュージ・ページ数の構成**:**

ヒュージ・ページはシステムの ハードウェア管理コンソール (HMC) で構成する必要があります。

1. 管理対象システムで、「属性」 > 「メモリー」 > 「詳細オプション」 > 「詳細の表示」 の順に進み、16 GB ページ数を変更します。
2. パーティションのプロファイルを変更して、16 GB ヒュージ・ページをパーティションに割り当てます。

複数ページ・サイズ・アプリケーションのサポート

32 ビット・プロセスまたは 64 ビット・プロセスのアドレス・スペースの 4 つの領域に対して使用する、複数のページ・サイズを指定できます。

これらのページ・サイズは環境変数を使用して構成するか、あるいは、以下のように **ldedit** または **ld** コマンドでアプリケーションの XCOFF バイナリーの設定値を使用して構成できます。

領域	**ld** または **ldedit** オプション	**LDR_CNTRL** 環境変数	説明
データ	-bdatapsize	DATAPSIZE	初期化データ、bss、ヒープ
スタック	-bstackpsize	STACKPSIZE	初期スレッド・スタック
テキスト	-btextpsize	TEXTPSIZE	主実行可能テキスト
共有メモリー	なし	SHMPSIZE	プロセスで割り当てられた共有メモリー

プロセスのアドレス・スペースの 4 つの領域ごとに、異なるページ・サイズを指定して使用することができます。両方のインターフェースとも、ページ・サイズはバイト数で指定してください。 指定されたページ・サイズはサイズ単位を示すサフィックスを付けて限定できます。 サポートされるサフィックスは、以下のとおりです。

- K (キロバイト)
- M (メガバイト)
- G (ギガバイト)

これらは大/小文字のいずれでも指定できます。

4 KB および 64 KB ページ・サイズは、4 つのすべてのメモリー領域に対してサポートされます。 16 MB ページ・サイズは、プロセス・データ、プロセス・テキスト、およびプロセス共有メモリー領域に対してのみサポートされます。 16 GB ページ・サイズは、プロセス共有メモリー領域に対してのみサポートされます。

非デフォルト・ページ・サイズを選択することにより、同一セグメントで選択されたページ・サイズより小さいページ・サイズの使用を明示的に使用不可にします。

サポートされないページ・サイズが指定されると、カーネルはその次の最小サポート・ページ・サイズを使用します。 指定ページ・サイズより小さいページ・サイズがない場合、カーネルは 4 KB ページ・サイズを使用します。

SHMPSIZE 環境変数を使用してプロセスの共有メモリーに使用するページ・サイズを指定するためのサポートが使用できます。以前のバージョンの AIX では、SHMPSIZE 環境変数は無視されます。 SHMPSIZE 環境変数は、プロセスが **shmget** サブルーチン、**ra_shmget** サブルーチン、および **ra_shmgetv** サブルーチンを呼び出したときに、そのプロセスで作成された System V 共有メモリー領域のみに適用されます。 SHMPSIZE 環境変数は、EXTSHM 共有メモリー領域と POSIX リアルタイム共有メモリー領域には適用されません。プロセスの SHMPSIZE 環境変数は、プロセスが他のプロセスで作成された共有メモリー領域を使用するため、共有メモリー領域には適用されません。

ldedit または **ld** コマンドによるアプリケーションの優先ページ・サイズの設定**:**

アプリケーションの XCOFF/XCOFF64 バイナリーに優先ページ・サイズを設定するには、**ldedit** または **ld** コマンドを使用します。

実行可能プログラムへリンクする場合のこれらのページ・サイズ・オプションの設定は、**ld** または **cc** コマンドを使用します。

```
ld –o mpsize.out –btextpsize:4K –bstackpsize:64K sub1.o sub2.o
cc –o mpsize.out –btextpsize:4K –bstackpsize:64K sub1.o sub2.o
```

既存の実行可能プログラムにこれらのページ・サイズ・オプションを設定する場合は、**ldedit** コマンドを使用します。

```
ldedit -btextpsize=4K -bdatapsize=64K -bstackpsize=64K mpsize.out
```

注**: ldedit** コマンドには、等号符号 (=) で指定されたページ・サイズ・オプションの値が必要ですが、**ld** および **cc** コマンドには、コロン (:) で指定されたページ・サイズ・オプションの値が必要です。

環境変数によるアプリケーションの優先ページ・サイズの設定**:**

プロセスの優先ページ・サイズは *LDR_CNTRL* 環境変数を使用して設定できます。

例えば、以下のコマンドでは、`mpsize.out` プロセスは、サポート・ハードウェアでデータには 4 KB ページ、テキストには 64 KB ページ、スタックには 64 KB ページ、および共有メモリーには 64 KB ページを使用します。

```
$ LDR_CNTRL=DATAPSIZE=4K@TEXTPSIZE=64K@SHMPSIZE=64K mpsize.out
```

実行可能プログラムの XCOFF ヘッダーに設定されたページ・サイズは、ページ・サイズの環境変数でオーバーライドされます。 さらに、いずれの *LARGE_PAGE_DATA* 環境変数の設定も *DATAPSIZE* 環境変数でオーバーライドされます。

複数ページ・サイズ・アプリケーションの考慮事項:

32 ビット・プロセス、スレッド・スタック、共用ライブラリー、またはラージ・ページ・データに関する問題は、複数ページ・サイズをサポートする AIX の能力に影響を及ぼすことがあります。

32 ビット・プロセス

デフォルトの AIX 32 ビット・プロセスのアドレス・スペース・モデルを使用して、プロセスの初期スレッド・スタックおよびデータは同じ PowerPC® 256 MB セグメントに配置されます。 現在は、1 つのセグメントには 1 種類のページ・サイズのみ使用できます。 したがって、標準 32 ビット・プロセスのスタックおよびデータに複数の異なるページ・サイズを指定すると、小さいほうのページ・サイズが使用されることになります。

大規模および超大規模プログラム・サポートの場合、このスタック以外のセグメントに 1 個のプロセスのデータ・ヒープを配置する代替アドレス・スペース・モデルのうちの 1 つを使用することにより、32 ビット・プロセスで、この初期スレッド・スタックおよびデータに複数の異なるページ・サイズを使用できます。

スレッド・スタック

デフォルトで、マルチスレッド・プロセスのスレッド・スタックは、プロセスのデータ・ヒープから取り込まれます。 したがって、マルチスレッド・プロセスでは、スタック・ページ・サイズの設定はプロセスの初期スレッドのためのスタックのみに適用されます。 この後に続くスレッドのためのスタックはプロセスのデータ・ヒープから割り当てられ、これらのスタックにはデータ・ページ・サイズの設定に指定されたサイズのページが使用されます。

さらに、マルチスレッド・プロセスのデータ用に 4 KB ページではなく 64 KB ページを使用すると、プロセスがスタック保護ページに必要な位置合わせのために作成することができるスレッドの最大数を削減できます。 この制限を超えてしまうアプリケーションでは、スタック保護ページを使用不可にして *AIXTHREAD_GUARDPAGES* 環境変数を 0 に設定し、さらにスレッドを作成できるようにします。

共用ライブラリー

64 KB ページをサポートするシステムでは、AIX はパフォーマンスを改善するためのグローバル共用ライブラリー・テキスト領域として 64 KB ページを使用できます。

ラージ・ページ・データ

DATAPSIZE 環境変数は *LARGE_PAGE_DATA* 環境変数に優先します。 さらに、アプリケーションの XCOFF バイナリーの *DATAPSIZE* 設定は、同じバイナリーにおけるいずれの lpdata 設定よりも優先します。

可変ラージ・ページ・サイズ・サポート

IBM Power Systems プロセッサー・ベースのサーバーでは、この概念が拡大され、単一セグメント内で 4 KB、64 KB、および 16 MB のページ・サイズの混合がサポートされます。

AIX オペレーティング・システムは、ハイパフォーマンス環境を改善するために 16 MB ページの使用をサポートしますが、メモリー・ページに柔軟性がなく、管理しやすくもありません。16 MB ページはページアウトされることができず、また新しい 16 MB ページは自動的に作成されることができません。

IBM Power Systems プロセッサー・ベースのサーバーは 16 MB の混合ページをサポートします。その結果、オペレーティング・システムは 4 KB または 64 KB の細分性のあるメモリー管理を柔軟に行えるよ

うになり、アプリケーションは 16 MB のハードウェア・ページ変換を使用してメモリーにアクセスできるようになります。 AIX オペレーティング・システムでのこのハードウェア・フィーチャーの使用を可変ラージ・ページ・サイズ・サポート (VLPSS) と呼びます。

VLPSS では、16 MB のサイズの、物理的に連続した 4 KB または 64 KB のページのブロックに位置合わせされたユーザー・メモリー領域が、配置されます。これらのメモリー・ページは単一の 16 MB 変換を通してアクセスできます。単一の 16 MB ページ変換を使用するため、その下にある 4 KB および 64 KB のページは同一のページ属性を持つ必要があり、メモリーに常駐する必要があります。ページ属性には、読み取り/書き込みページ保護、ストレージ・キー保護、および非実行保護があります。

16 MB VLPSS ページは、オペレーティング・システムにより、16 MB 変換から元の 4 KB または 64 KB のページ・サイズ変換にデモートされることがあります。ページがデモートされるのは、メモリーの一部をページング・デバイスにページアウトすることをオペレーティング・システムが必要とする場合、またはアプリケーションによる 16 MB 領域のページ属性の変更のために属性が均一でなくなる場合です。本来の 16 MB ページにはこのような柔軟性がありません。

アプリケーションは、vm_pattr システム・コールを使用し、**VM_PA_SET_PSIZE_EXTENDED** コマンドを指定することにより、VPLSS フィーチャーの利点を活用できます。オペレーティング・システムは、vm_pattr システム・コールのアドバイスを受け入れることができ、システムが影響される場合はアドバイスを拒否することもできます。

16 MB ページ・サイズは、64 KB 動的可変ページ・サイズと比べて、より大きな量の連続メモリーとなります。 VLPSS を使用するようにメモリーを配置してプロモートすることは費用のかかる操作であり、システム全体のパフォーマンスに好ましくない影響が出ることがあります。したがって、メモリーを 16 MB ページ・サイズにプロモートすることには制約がありますが、これは動的可変ページ・サイズ・サポートにはない制約です。

VLPSS フィーチャーの使用は、CAP_BYPASS_RAC_VMM および CAP_PROPAGATE 機能を持つユーザーまたは root 権限を持つユーザーに限定されます。

16 MB ユーザー・メモリー領域は、VLPSS フィーチャーに用いられるためにメモリーに完全に常駐する必要があります。オペレーティング・システムは VLPSS フィーチャーを使用するために大量のシステム・メモリーを必要とします。このフィーチャーに必要な最小メモリー・サイズは 16 GB です。

ページ・サイズおよび共有メモリー

shmctl() システム・コールには、**SHM_PAGESIZE** コマンドで System V 共有メモリーに使用するページ・サイズを選択できます。

非デフォルト・ページ・サイズを選択することにより、同一セグメントで選択されたページ・サイズより小さいページ・サイズの使用を明示的に使用不可にします。

共有メモリー領域用のページ・サイズを選択するための shmctl() の使用について詳しくは、*Technical Reference: Base Operating System and Extensions, Volume 2*の『shmctl()』トピックを参照してください。

ps コマンドによるプロセスのページ・サイズの判別

プロセスのデータ、スタック、およびテキストに使用されているベース・ページ・サイズをモニターするには、**ps** コマンドを使用することができます。

次に、**ps -Z** コマンドの出力例を示します。 *DPGSZ* 欄にはプロセスのデータ・ページ・サイズが表示されます。 *SPGSZ* 欄にはプロセスのスタック・ページ・サイズが表示されます。 *TPGSZ* 欄にはプロセスのテキスト・ページ・サイズが表示されます。

```
# ps -Z
    PID    TTY  TIME DPGSZ SPGSZ TPGSZ CMD
 311342  pts/4  0:00    4K    4K    4K ksh
 397526  pts/4  0:00    4K    4K    4K ps
 487558  pts/4  0:00   64K   64K    4K sleep
```

vmstat コマンドを使用したページ・サイズ・モニター

vmstat には、特定ページ・サイズのメモリー統計情報を表示するために使用可能な 2 つのオプションがあります。

vmstat -p
ページ・サイズごとの統計情報の明細に加えて、グローバル vmstat 情報も表示します。

vmstat -P
ページ・サイズごとの統計情報を表示します。

両方のオプションとも特定ページ・サイズのコンマ区切りリストを取り込むか、または 1 つ以上のページ・フレームを持つすべてのサポートされるページ・サイズについて、情報の表示を指示するキーワード **all** を取り込みます。 以下の例では、システム上のページ・フレームを持つすべてのページ・サイズについて、ページ・サイズごとの情報を表示しています。

```
# vmstat -P all
System configuration: mem=4096MB
pgsz            memory                           page
----- -------------------------- ------------------------------------
           siz      avm      fre    re    pi    po    fr     sr    cy
   4K   542846   202832   329649     0     0     0     0      0     0
  64K    31379      961    30484     0     0     0     0      0     0
```

svmon コマンドを使用したシステム全体のページ・サイズ・モニター

システム全体でのページ・サイズの使用を表示する場合は、**svmon** コマンドを使用します。

svmon コマンドは拡張されて、統計情報のページ・サイズごとの明細を提供できるようになりました。例えば、各ページ・サイズのグローバル統計情報を表示する場合は、**-G** オプションを指定した **svmon** コマンドを使用します。

```
# svmon -G
               size      inuse       free        pin    virtual
memory      8208384    5714226    2494158     453170    5674818
pg space     262144      20653

               work       pers       clnt
pin          453170          0          0
in use      5674818        110      39298

PageSize   PoolSize      inuse       pgsp        pin    virtual
s   4 KB          -    5379122      20653     380338    5339714
m  64 KB          -      20944          0       4552      20944
```

詳しくは、*Commands Reference, Volume 5*の『**svmon** コマンド』を参照してください。

ラージ・ページ・サイズのためのメモリー使用の考慮事項

ユーザー・アプリケーションでラージ・ページ・サイズの使用について、可能性のあるパフォーマンスへの影響を評価する場合は、ワークロードのメモリー使用を考慮する必要があります。

ラージ・ページ・サイズを使用すると、メモリーのフラグメント化のためにワークロードのメモリー占有スペースが増加することがあります。 ラージ・ページ・サイズの使用によりワークロードのメモリー占有スペースが増加するかどうかを判別するには、**svmon** および **vmstat** コマンドを使用して、ワークロードのメモリー使用をモニターすることができます。

64 KB ページ・サイズについて検討している場合は、代わりにデフォルトのページ・サイズを使用し、オペレーティング・システムにどちらのページ・サイズを使用するかを決定させてください。 64KB の全領域が必要であるなど、アプリケーションの実効ページ・セットが高密度の場合は、メモリーが (割り当てられていても使用されないなどで) 無駄になることがほとんどないので、64 KB ページを選択するのが適切です。

連続メモリー最適化

連続メモリー最適化プログラムは、ページの統合とプロモーションを動的に実行します。

動的可変ページ・サイズのサポート

参照回数がわずかなメモリー領域での 4 KB ページ・サイズの保守的なメモリー使用率を、参照回数が多いメモリー領域での 64 KB ページ・サイズのパフォーマンス上の利点と組み合わせることを、ユーザー介入なしに自動的に行う AIX バージョン 6.1 の機能は、動的可変ページ・サイズのサポート (DVPSS) と呼ばれます。

DVPSS は、単一セグメント内での混合ページ・サイズをサポートする POWER6 の機能に基づいています。 このアーキテクチャーは、異なるページ・サイズのさまざまな置換をサポートします。ただし、POWER6 では 4 KB と 64 KB のページ・サイズの組み合わせがサポートされます。

可変ページ・サイズ・セグメントのデフォルト設定は、16 すべての 4 KB ページが参照されるまで、64 KB サイズで位置合わせされた領域内の 4 KB ページと 4KB 変換です。 16 ページすべてが参照されると、DVPSS は、同じ状態であることを確認するために状態を検査します (同一のページ読み取りまたは書き込みの保護、非実行保護、ストレージ・キー保護、および入出力状態になっていないことなど)。 確認が済むと、4 KB 変換が除去されて、64 KB 変換に置き換えられます。

Continuous Program Optimization Agent (CPOagent)

オペレーティング・システムによる DVPPS 方式のサポートにおける制限は、ページ・サイズが 64 KB にプロモートされる前に 16 の 4 KB ページがすべて参照されなければならないことです。 CPOagent は、ページの統合とプロモーションを動的に実行する連続メモリー最適化を使用して、この制限を克服するのに役立ちます。 このフィーチャーは、AIX 6.1 テクノロジー・レベル 6 以上に適用されます。

CPOagent には以下でアクセスできます。

```
usr/lib/perf/CPOagent
```

構文

CPOagent [-f *configuration file*]

フラグ

項目	説明
-f	デフォルトの構成ファイル名を変更します。 このオプションが指定されない場合、ファイル名は /usr/lib/perf/CPOagent.cf ファイルの場所で入手可能であると想定されます。

デフォルトでは、CPOagent は実行されません。 root ユーザーが CPOagent を明示的に開始できます。CPOagent が開始すると、バックグラウンドで実行され、より大きいページ・サイズを使用する利点がある候補プロセスを識別します。 候補プロセスは、指定されたしきい値を超えるメモリーとプロセッサーの使用量に基づいて識別されます。

注: CPOagent は現在、64 KB までページ・サイズをプロモートできます。

CPOagent 構成ファイル

CPOagent が開始すると、構成ファイル内の情報を読み取って解析します。 構成ファイルは、以下のものを含むフィールドを持つスタンザ・ファイルです。

```
TCPU=<n1>
TMEM=<n2>
PATI=<n3>
PATM=<n4>
PPTS=<n5>
TOPM=<n6>
PFLR=<c>
```

構成ファイル内のフィールドの説明は次のとおりです。

フィールド	説明
TCPU	プロセスごとの CPU 使用量しきい値をパーセンテージで指定します。 デフォルト: 25 最小: 10 最大: 100
TMEM	プロセスごとのメモリー使用量しきい値を MB で指定します。 デフォルト : 1 最小: 1
PATI	ページ分析時間間隔 (PATI) を分単位で指定します。統合され、より高いサイズにプロモートできるページを識別するために、候補プロセスが分析される時間間隔を指定します。 デフォルト: 15 最小: 5 最大: 60
PATM	ページ分析時間モニター (PATM) を秒単位で指定します。ページ統合とプロモーションのための候補ページを識別するために、ページ使用量統計が収集される時間を指定します。 デフォルト: 30 最小: 5 最大: 180

フィールド	説明
PPTS	ページ・プロモーション・トリガー・サンプル (PPTS) を指定します。 ページ・プロモーションをトリガーする前に収集されるサンプル数を指定します。 デフォルト: 4 最小: 4
TOPM	ページの統合とプロモーションのために検討が必要な、CPU ごとの上位 CPU 消費プロセス数を指定します。 デフォルト: 2 最小: 1
PFLR	ワイルドカードおよびワイルドカードと一致するプロセスが、ページの統合とプロモーションのために CPOagent によって検討されることを指定します。 プロセス・フィルター・ワイルドカード (PFLR) と呼ばれます。

CPOagent を使用する利点

CPOagent メカニズムを使用する利点には、次のものがあります。

- アプリケーションは、変更に対して透過的です。 したがって、アプリケーションに変更を加える必要はありません。
- ページ・プロモーションは、CPOagent.cf ファイルで管理者によって設定されるポリシーに基づいて実行されます。これは、CPU リソースとメモリー・リソースに対するワークロードの需要によって異なります。 これは、効率的なページ・プロモーションに役立ちます。 このプロセスにより、動的な可変ページ・サイズ・サポートに対するきめ細かい制御が可能になり、要件のあるアプリケーションに対してページ・プロモーションが実行されることが確実になります。

サンプル・シナリオ

システムで実行されるアプリケーション StressEngine について検討します。 このアプリケーションには、高い CPU 消費量とメモリー消費量があります。 CPOagent がないと、StressEngine アプリケーションは、特定セグメントの 16 ページすべてが参照され、ページが同じ状態になるまで、動的可変ページ・サイズ・サポートを利用できません。ページ・サイズは、**svmon** コマンドで生成されるプロセス・レポートで確認できます。

```
# svmon -P 8454254 -O pgsz=on,unit=MB,segment=on
Unit: MB
--------------------------------------------------------------------------------
     Pid Command          Inuse      Pin     Pgsp  Virtual
 8454254 StressEngine   157.87     42.3        0   157.84

     PageSize                Inuse        Pin       Pgsp    Virtual
     s    4 KB                64.2       0.02          0       64.2
     m   64 KB                93.7       42.3          0       93.7

    Vsid      Esid Type Description              PSize  Inuse   Pin Pgsp Virtual
  86f49b         2 work process private             s   64.1  0.02    0    64.1
    9000         d work shared library text         m   47.9     0    0    47.9
    8002         0 work fork tree                   m   45.8  42.3    0    45.8
                   children=939b1c, 0
  80fdc3         f work shared library data         s   0.09     0    0    0.09
  85fd37         1 clnt code,/dev/hd1:10            s   0.02     0    -       -
```

CPOagent が開始し、次のサンプル CPOagent.cf ファイルがあるとします。

```
-----------------------------------------
TCPU=25
TMEM=50
```

```
PATI=15
PATM=30
PPTS=4
TOPM=2
PFLR=Stress*
-------------------------------------------
```

構成ファイルに応じて、CPOagent は 15 分間巡回します (PATI =15)。 特定の 15 分間隔の間、実行中のプロセスの CPU およびメモリー使用量をモニターします。プロセス名 Stress を持ち (PFLR = Stress*)、CPU 使用量が 25% を超え (TCPU = 25)、メモリー使用量が 50 MB を超える (TMEM = 50)、上位 2 つのプロセス (TOPM =2) が、ページ統合とプロモーションの候補になります。 このプロセスは、ページ統合とプロモーションのためにアルゴリズムをトリガーする前に、4 つのサンプルを収集して (PPTS = 4) 検証します。 さらに、ページ統合とプロモーション用の候補ページを識別するために、ページ使用量統計が 30 秒間収集されます (PATM =30)。 CPOagent が実行しているので、特定セグメントの 16 ページすべてが参照されるのを待機しません。 CPOagent は、CPOagent.cf 構成ファイル、および CPU リソースとメモリー・リソースに対するアプリケーションの需要を参照することによって、アプリケーションでページ統合とプロモーションが必要かどうかを評価します。 プロモートされるページは、**svmon** コマンドで生成されるプロセス・レポートから明らかになります。

```
# svmon -P 8454254 -O pgsz=on,unit=MB,segment=on
Unit: MB
--------------------------------------------------------------------------------
     Pid Command          Inuse      Pin     Pgsp  Virtual
 8454254 StressEngine    157.87     42.3        0   157.84

     PageSize                Inuse        Pin       Pgsp    Virtual
     s    4 KB                64.2       0.02          0       64.2
     m   64 KB                93.7       42.3          0       93.7

    Vsid      Esid Type Description              PSize  Inuse   Pin Pgsp Virtual
  86f49b         2 work process private            sm   64.1   0.02    0    64.1
    9000         d work shared library text         m   47.9      0    0    47.9
    8002         0 work fork tree                   m   45.8   42.3    0    45.8
                   children=939b1c, 0
  80fdc3         f work shared library data       sm   0.09      0    0    0.09
  85fd37         1 clnt code,/dev/hd1:10           s   0.02      0    -       -
```

VMM スレッド中断オフロード

VMM スレッド中断オフロード (VTIOL) インフラストラクチャーは、VMM が iodone() サービスの処理をカーネル・スレッドにオフロードできるようにします。

VTIOL 機能は、iodone() プロセスが高優先順位スレッドのパフォーマンスに影響する可能性を低くするために使用します。 VTIOL 機能は、高優先順位プロセスを中断するのではなく、バックグラウンド・スレッドを使用して iodone() サービスを扱います。 VMM は、iodone() サービスの処理をオフロードするかどうかを決定するための発見的手法をいくつか使用します。例えば、バックグラウンド書き込み操作や先読み操作など、明示的な waiter スレッドのない特定の入出力操作はオフロード可能です。明示的な waiter スレッドのある入出力操作は、その入出力操作がもっと高い優先順位で実行する必要があることを示す場合があります。そのような場合、入出力操作はオフロードされず、割り込みレベルで処理されます。

VMM スレッド中断オフロードの優先順位変更

VMM スレッド中断オフロードの優先順位変更 (VTIOLR) インフラストラクチャーは、VMM スレッド中断オフロード (VTIOL) 機能を向上させます。

VTIOLR インフラストラクチャーにより、AIX オペレーティング・システムはオフロードされた VMM 入出力操作が処理される順序を、指定の基準に基づいて優先順位付けることができます。例えば、ファイル

システム sync() プロセスからの大規模な入出力実行操作の後に小規模な read() 操作がキューに入れられている場合、read() 操作がフォアグラウンド・ジョブと見なされる一方で、sync() プロセスがバックグラウンド・ジョブと見なされる可能性があります。この場合、VTIOLR 機能は read() 操作の優先順位を変更し、この入出力操作を実行してから、その他の iodone() バッファーを処理することができます。優先順位が変更された iodone() バッファーは、それらを一連のバックグラウンド・スレッドにオフロードすることによって処理されます。したがって、入出力操作はバックグラウンド・スレッドによって実行され、割り込みレベルでは実行されません。

論理ボリュームおよびディスク入出力のパフォーマンス

このトピックでは、論理ボリュームとローカル接続ディスク・ドライブのパフォーマンスを取り上げます。

オペレーティング・システムのボリューム・グループ、論理ボリュームと物理ボリューム、またはロジカル・パーティションと物理パーティションの概念に慣れていない方は、『ハード・ディスク・ストレージ管理のパフォーマンスの概要』を参照してください。

パフォーマンス上の理由から、ハード・ディスクの数とタイプ、およびそのハード・ディスク上のページ・スペースと論理ボリュームのサイズと配置の決定は、プリインストールの重要なプロセスです。 プリインストールのディスク構成計画に関する考慮事項についての広範なディスカッションについては、『ディスクのプリインストールのガイドライン』を参照してください。

関連概念:

106 ページの『ディスクのプリインストールのガイドライン』

論理ボリュームを定義し拡張するためのメカニズムは、可能な最善のデフォルト選択を行うようになっています。 しかし、満足のいくディスク入出力パフォーマンスが得られる可能性は、システムのインストーラーが論理ボリュームのサイズと配置を、予期されるデータ・ストレージおよびワークロード要件に合わせて調製した場合の方がはるかに高くなります。

ディスク入出力のモニター

ディスク入出力をモニターする場合は、一連のアクションを決定するために考慮すべき幾つかの問題があります。

- 最もアクティブなファイル、ファイルシステム、および論理ボリュームを検索する。
  - 「ホット」なファイルシステムを、物理ドライブ上でさらに良く配置することができるか、または複数の物理ドライブに分散して配置できるか。 (**lslv**、**iostat**、**filemon**)
  - 「ホット」なファイルはローカルかリモートか。 (**filemon**)
  - ページング・スペースがディスク使用率の大半を占めるか。 (**vmstat**、 **filemon**)
  - プロセスの実行によって使用されているファイル・ページのキャッシュに十分なメモリーがあるか。 (**vmstat**、**svmon**)
  - アプリケーションは大量の同期 (キャッシュを使用しない) ファイル入出力を実行するか。
- ファイルのフラグメント化を判別する。
  - 「ホット」なファイルは過度にフラグメント化されているか。 (**fileplace**)
- 使用率が最高の物理ボリュームを検出する。
  - ドライブまたは入出力アダプターのタイプがボトルネックの原因になっていないか。 (**iostat**、**filemon**)

事前チューニング・ベースラインの準備

ディスク構成を大幅に変更したりパラメーターをチューニングしたりする前に、現在の構成とパフォーマンスを記録する、測定のベースラインを作成しておくとよいでしょう。

入出力待ち時間の報告

AIX バージョン 6.1 以降では、ディスク入出力の待機に費やすプロセッサー時間 (wio 時間) のパーセンテージの計算に使用する方式が拡張されています。

アイドル CPU で未解決の入出力が開始された場合、その CPU には wio というマークが付けられます。

また、マウントされた NFS ファイルシステムの入出力の待機が、入出力待ち時間として報告されるようになりました。

iostat コマンドによるディスク・パフォーマンスの評価

評価は、システムのワークロードのピーク期間中、または入出力遅延を最小にする必要がある重要なアプリケーションの実行中に、interval パラメーターを指定して **iostat** コマンドを実行することによって開始します。

次のシェル・スクリプトは、測定すべき入出力があるように、ラージ・ファイルのコピーをフォアグラウンドで実行している間に、バックグラウンドで **iostat** コマンドを実行します。

```
# iostat 5 3 >io.out &
# cp big1 /dev/null
```

AIX オペレーティング・システムはディスク・アクティビティーのヒストリーを維持します。ディスク入出力のヒストリーが使用不可にされていると、**iostat** コマンドを実行した場合に以下のメッセージが表示されます。

```
Disk history since boot not available.

The interval disk I/O statistics are unaffected by this.
```

ディスク入出力のヒストリーを使用可能にするには、コマンド・ラインから `smit chgsys` と入力し、「ディスク入出力のヒストリーを継続して維持する **(Continuously maintain DISK I/O history)**」フィールドで「はい **(true)**」を選択します。

以下の例では、3 つのレポートが `io.out` ファイル内に格納されます。

```
# iostat 5 3 >io.out &
# cp big /dev/null

System configuration: lcpu=4 drives=1 ent=0.50 paths=1 vdisks=1

tty:      tin         tout    avg-cpu: % user % sys % idle % iowait physc % entc
          0.0          0.0      0.1      0.6    99.2   0.1    0.0     1.2

Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk0           0.4       5.6       0.6         28         0

tty:      tin         tout    avg-cpu: % user % sys % idle % iowait physc % entc
          0.0          0.0      0.1      1.4    98.4   0.2     0.0    2.4

Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk0           6.0     123.2      10.6        4       612

tty:      tin         tout    avg-cpu: % user % sys %  idle % iowait physc % entc
          2.6          2.6     0.1        0.5   99.4    0.0    0.0     1.1
```

```
Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk0           0.0       0.0       0.0          0         0
```

最初のレポートは、最終リブート以後の要約で、入出力における各ハード・ディスクの全体的なバランス (この場合は、不均衡) を示します。 hdisk1 はほとんどアイドルで、hdisk2 は総入出力の約 63% を受け持ちました (Kb_read と Kb_wrtn から)。

2 番目のレポートは、**cp** が実行された 5 秒の間隔を示しています。この **cp** の経過時間は約 2.6 秒でした。 したがって、入出力依存性の高い 2.5 秒は、2.5 秒のアイドル時間によって平均化されて、39.5% の「% iowait」が報告されています。 間隔がもっと短ければ、コマンド自体の特徴をより詳細に表現できるはずですが、これは、複数の間隔にまたがる平均アクティビティーを示すレポートを見る場合に、何に注意すべきかを示すための実例です。

関連概念:

115 ページの『iostat コマンド』
iostat コマンドは、システムに、ディスク入出力の制約によるパフォーマンス上の問題があるかどうかに関して、最初の印象を得る最も速い方法です

関連資料:

16 ページの『iostat コマンドを使用した連続パフォーマンス・モニター』
iostat コマンドは、ディスクおよび CPU の使用状況を判別するために便利です。

TTY レポート:

iostat の出力の TTY 情報の 2 つの欄 (*tin* および *tout*) は、すべての TTY デバイスによって読み書きされた文字数を示しています。

これには、実 TTY デバイスと疑似 TTY デバイスの両方が含まれます。 実 TTY デバイスは、非同期ポートに接続されているものです。 一部の疑似 TTY デバイスは、シェル、**telnet** セッション、および **aixterm** ウィンドウです。

入出力文字の処理には CPU リソースを消費するので、TTY アクティビティーの増加と CPU 使用率の間の相関を探します。そのような関係が存在するなら、TTY サブシステムのパフォーマンスを改善する方法を評価します。 実行できるステップには、アプリケーション・プログラムの変更、ファイル転送時の TTY ポート・パラメーターの変更、またはおそらくより高速または効率のよい非同期式通信アダプターへのアップグレードなどがあります。

マイクロプロセッサー・レポート:

マイクロプロセッサー統計情報欄 (% user、% sys、% idle、および % iowait) により、マイクロプロセッサー使用率の明細が提供されます。

この情報は、**vmstat** コマンドの出力でも、「us」、「sy」、「id」、および「wa」というラベルのついた欄に報告されます。これらの値の詳細については、 112 ページの『vmstat コマンド』を参照してください。さらに、 194 ページの『入出力待ち時間の報告』で説明する、「% iowait」に加えられた変更についても注意してください。

1 つのアプリケーションを実行しているシステムで、入出力待ちのパーセンテージが高い場合は、ワークロードに関連している可能性があります。 多くのプロセスがあるシステムでは、一部が入出力待ちしている間に、他のプロセスが実行します。この場合は、実行中のプロセスが待ち時間の一部を「隠す」ので、「% iowait」が小さいかゼロになる場合があります。 「% iowait」は低いのですが、ボトルネックによってアプリケーションのパフォーマンスが制限される場合があります。

iostat コマンドがプロセッサー制約の状態になっていないことを示し、かつ、% iowait 時間が 20% を超えている場合は、I/O またはディスク制約の状態である可能性があります。 この状態は、実メモリーの不足のために過度のページングが行われたことが原因と考えられます。 また、ディスク負荷、フラグメント化されたデータまたは使用パターンの不均衡による場合もあります。 ディスク負荷の不均衡の場合、同じ **iostat** レポートで必要な情報が提供されます。 しかし、論理リソースであるファイルシステムや論理ボリュームに関する情報については、**filemon** コマンドや **fileplace** コマンドなどのツールを使用する必要があります。

ドライブ・レポート**:**

ドライブ・レポートには物理ドライブに関するパフォーマンス関連の情報が含まれています。

ディスク入出力のパフォーマンス上の問題が疑われる場合は、**iostat** コマンドを使用します。 TTY および CPU の統計情報に関する情報を除外するには、**-d** オプションを使用します。さらに、ディスク統計情報は、ディスク名を指定することによって、重要なディスクに限定することができます。

最初のデータの集合は、システム起動以降のすべてのアクティビティーを表していることを覚えておいてください。

Disks:
物理ボリュームの名前を示します。 これらは、hdisk または cd のどちらかの後に番号を付けたものです。 物理ボリューム名を **iostat** コマンドで指定した場合、指定した名前だけが表示されます。

% tm_act
物理ディスクがアクティブだった時間のパーセンテージ (ドライブの帯域幅使用率)、言い換えれば、ディスク要求が未解決だった時間の合計を示します。 ドライブがアクティブなのは、データ転送および新しいロケーションへのシークなどのコマンド処理の間です。 「ディスク・アクティブ時間 (disk active time)」のパーセンテージは、リソースのコンテンションに比例しており、パフォーマンスに逆比例します。ディスクの使用が増加するにつれて、パフォーマンスが減少し、応答時間が増えます。一般に、使用率が 70% を超えると、プロセスは、入出力の完了に必要な時間より長く待つことになります。大部分の UNIX プロセスでは、入出力要求が完了するのを待っている間はブロック (またはスリープ) するからです。 使用中ドライブとアイドル・ドライブを探します。 使用中ドライブからアイドル・ドライブへデータを移動すると、ディスクのボトルネックの緩和に役立つことがあります。 ディスクとの間のページングは、入出力負荷に寄与します。

Kbps ドライブに転送 (読み取りまたは書き込み) されたデータの量 (KB/ 秒) を示します。 これは Kb_read と Kb_wrtn の合計を、レポート作成間隔の秒数で割った値です。

tps 物理ディスクに対して出された、1 秒当たりの転送の数を示します。 転送は、デバイス・ドライバー・レベルでの物理ディスクに対する入出力要求です。 複数の論理要求は、ディスクに対する単一の入出力要求として結合することができます。 転送は、中間のサイズです。

Kb_read
測定した間隔中に物理ボリュームから読み取ったデータの合計 (KB 単位) を報告します。

Kb_wrtn
測定した間隔中に物理ボリュームに書き込んだデータの合計 (KB 単位) を示します。

統計情報はアプリケーションの特性、システム構成、およびディスク・ドライブとアダプターのタイプに非常に緊密に関係しているので、単独で考えた場合、上記のフィールドにはいずれも受け入れられない値はありません。 したがって、データを評価するときは、パターンと関係を調べてください。 最もよく見られる関係は、ディスク使用率 (%tm_act) とデータ転送速度 (tps) の間の関係です。

このデータから何らかの有効な結論を導き出すには、システムの物理ディスク・ドライブおよびアダプターのタイプばかりでなく、順次、ランダム、あるいは両方の組み合わせなどの、アプリケーションがディスクのデータにアクセスするパターンを理解する必要があります。 例えば、アプリケーションが順次に読み取り/書き込みを行う場合、ディスクのビジー率 (%tm_act) が高いときは、ディスク転送速度 (Kbps) が高いと予測する必要があります。 「Kb_read」および「Kb_wrtn」の欄は、アプリケーションの読み取り/書き込みの動作の理解を確認することができます。 ただし、これらの欄からは、データ・アクセスのパターンは得られません。

一般に、ディスク転送速度 (Kbps) が高いかぎり、ディスクのビジー率 (%tm_act) の高さに関して心配する必要はありません。 ただし、ディスク・ビジー率が高くてディスク転送速度が低い場合は、論理ボリューム、ファイルシステム、または個々のファイルがフラグメント化されている可能性があります。

ディスク、論理ボリュームおよびファイルシステムのパフォーマンスについてのディスカッションでは、時として、システムのドライブが多いほど、ディスク入出力のパフォーマンスが良くなるという結論に達します。 ただし、ディスク・アダプターで扱うことができるデータの量に制限があるので、常に成り立つとは限りません。 ディスク・アダプターがボトルネックになる場合もあります。 すべてのディスク・ドライブが 1 つのディスク・アダプターにあり、ホット・ファイルシステムが別々の物理ボリュームにある場合、複数のディスク・アダプターを使用した方が良い場合があります。 パフォーマンスの改善はアクセスのタイプに左右されます。

特定アダプターが飽和しているかどうかを調べるには、**iostat** コマンドを使用して、 1 つのディスク・アダプターに接続されたディスクの Kbps の量をすべて合計します。 総合パフォーマンスを最大にするには、転送速度 (Kbps) の合計が、ディスク・アダプターのスループット・レーティングより下でなければなりません。 ほとんどの場合、スループット・レートの 70% を使用します。 AIX オペレーティング・システムでは、**-a** または **-A** オプションは、この情報を表示します。

vmstat コマンドによるディスク・パフォーマンスの評価

システムが入出力制約であることを証明するには、**iostat** コマンドを使用した方が良いでしょう。

ただし、**vmstat** コマンドでも、 112 ページの『vmstat コマンド』で説明したように、「*wa*」欄を見ればその方向を示すことができました。入出力制約のその他のインディケーターには、以下のものがあります。

- **vmstat** の出力の「disk xfer」部分

  論理ディスクに関する統計を表示するには (最大 4 つのディスクを表示可能)、次のコマンドを使用します。

```
# vmstat hdisk0 hdisk1 1 8
kthr     memory             page              faults        cpu     disk xfer
----  ----------  ------------------------ ------------ ----------- ------
r  b   avm   fre  re  pi  po  fr  sr  cy  in   sy  cs  us sy id wa  1 2 3 4
0  0  3456 27743   0   0   0   0   0   0 131  149  28  0  1 99  0  0 0
0  0  3456 27743   0   0   0   0   0   0 131   77  30  0  1 99  0  0 0
1  0  3498 27152   0   0   0   0   0   0 153 1088  35  1 10 87  2  0 11
0  1  3499 26543   0   0   0   0   0   0 199 1530  38  1 19  0 80  0 59
0  1  3499 25406   0   0   0   0   0   0 187 2472  38  2 26  0 72  0 53
0  0  3456 24329   0   0   0   0   0   0 178 1301  37  2 12 20 66  0 42
0  0  3456 24329   0   0   0   0   0   0 124   58  19  0  0 99  0  0 0
0  0  3456 24329   0   0   0   0   0   0 123   58  23  0  0 99  0  0 0
```

  「*disk xfer*」部分では、サンプル間隔中に発生した、指定された物理ボリュームに対する 1 秒ごとの転送の数を提示します。 1 つから 4 つまでの物理ボリューム名を指定できます。 転送統計情報は、指定された各ドライブについて、指定された順番に提示されます。 このカウントは、物理デバイスに対する要求を表しています。 これは、読み取りまたは書き込みされたデータの量を暗黙に示しているわけではありません。 複数の論理要求は、1 つの物理要求に結合することができます。

- **vmstat** の出力の「*in*」欄

  この欄では、測定間隔内に監視された (1 秒当たりの) ハードウェアまたはデバイス割り込みの数を示します。 割り込みの例としては、ディスク要求の完了や 10 ミリ秒クロック割り込みがあります。 後者は 1 秒当たり 100 回発生するので、「*in*」フィールドは常に 100 より大きくなります。 しかし、**vmstat** コマンドでは、システム割り込みに関してさらに詳細な出力が提供されます。

- **vmstat -i** の出力

  -i パラメーターは、システム起動以降の、各デバイスによって行われた割り込みの数を表示します。しかし、interval と (オプションで) count パラメーターを追加すると、起動以降の統計は最初のスタンザだけに表示されます。すべての末尾スタンザは、スキャンされた間隔に関する統計です。

```
# vmstat -i 1 2
priority level    type   count module(handler)
    0       0   hardware     0 i_misc_pwr(a868c)
    0       1   hardware     0 i_scu(a8680)
    0       2   hardware     0 i_epow(954e0)
    0       2   hardware     0 /etc/drivers/ascsiddpin(189acd4)
    1       2   hardware   194 /etc/drivers/rsdd(1941354)
    3      10   hardware 10589024 /etc/drivers/mpsdd(1977a88)
    3      14   hardware 101947 /etc/drivers/ascsiddpin(189ab8c)
    5      62   hardware 61336129 clock(952c4)
   10      63   hardware 13769 i_softoff(9527c)
priority level    type   count module(handler)
    0       0   hardware     0 i_misc_pwr(a868c)
    0       1   hardware     0 i_scu(a8680)
    0       2   hardware     0 i_epow(954e0)
    0       2   hardware     0 /etc/drivers/ascsiddpin(189acd4)
    1       2   hardware     0 /etc/drivers/rsdd(1941354)
    3      10   hardware    25 /etc/drivers/mpsdd(1977a88)
    3      14   hardware     0 /etc/drivers/ascsiddpin(189ab8c)
    5      62   hardware   105 clock(952c4)
   10      63   hardware     0 i_softoff(9527c)
```

  注**:** 出力は、ハードウェアおよびソフトウェアの構成によって、システムごとに異なります (例えば、クロック割り込みは通常の **vmstat** の出力では「*in*」欄の下に含まれていますが、**vmstat -i** の出力では表示されない場合があります)。「*count*」欄の値が高いかどうかチェックして、このモジュールがそれほど多くの割り込みを実行しなければならない理由を調べてください。

sar コマンドによるディスク・パフォーマンスの評価

sar コマンドは、システムに関する統計データの収集に使用される、標準の UNIX コマンドです。

sar コマンドには多数のオプションがあり、キューイング、ページング、TTY、およびその他多数の統計情報を提供します。 **sar -d** オプションは、リアルタイムのディスク入出力統計情報を生成します。

```
# sar -d 3 3

AIX konark 3 4 0002506F4C00    08/26/99

12:09:50     device    %busy    avque    r+w/s   blks/s   avwait   avserv

12:09:53     hdisk0      1      0.0        0        5      0.0      0.0
             hdisk1      0      0.0        0        1      0.0      0.0
                cd0      0      0.0        0        0      0.0      0.0

12:09:56     hdisk0      0      0.0        0        0      0.0      0.0
             hdisk1      0      0.0        0        1      0.0      0.0
                cd0      0      0.0        0        0      0.0      0.0

12:09:59     hdisk0      1      0.0        1        4      0.0      0.0
             hdisk1      0      0.0        0        1      0.0      0.0
```

```
                cd0      0      0.0        0        0      0.0      0.0

Average       hdisk0     0      0.0        0        3      0.0      0.0
              hdisk1     0      0.0        0        1      0.0      0.0
                cd0      0      0.0        0        0      0.0      0.0
```

sar -d コマンドによってリストされるフィールドは、次のとおりです。

%busy
デバイスが転送要求のサービスで使用中だった時間の部分。 これは、 **iostat** コマンドのレポートの「%tm_act」欄と同じです。

avque その時間に未解決だったアダプターからデバイスへの要求の平均数。 デバイス・ドライバーのキューに追加の入出力操作がある可能性があります。 この数は、入出力ボトルネックが存在するか否かの良いインディケーターです。

r+w/s デバイスからまたはデバイスへの読み取り/書き込み転送の数。 これは、**iostat** コマンドのレポートの「tps」と同じです。

blks/s 512 バイト単位で転送されたバイト数。

avwait
サービスを待っているトランザクションの平均数 (キューの長さ)。 転送要求がキュー上でアイドル状態でデバイスを待っていた平均時間 (ミリ秒)。 この数は、現在報告されておらず、デフォルトで 0.0 と表示されています。

avserv
平均シーク当たりのミリ秒数。 デバイスに関する各転送要求 (シーク、回転待ち時間、およびデータ転送回数を含む) のサービスの平均時間 (ミリ秒単位)。 この数は、現在報告されておらず、デフォルトで 0.0 と表示されています。

lslv コマンドによる論理ボリュームのフラグメント化の評価

lslv コマンドは、他の情報とともに、論理ボリュームのフラグメント化を表示します。

論理ボリュームのフラグメント化を調べるには、次のような **lslv -l** *lvname* コマンドを使用します。

```
# lslv -l hd2
hd2:/usr
PV               COPIES        IN BAND       DISTRIBUTION
hdisk0           114:000:000   22%           000:042:026:000:046
```

「COPIES」の出力は、論理ボリューム hd2 にコピーが 1 つだけあることを示しています。「IN BAND」は、イントラポリシー、すなわち論理ボリュームの属性にいかに良く従っているかを示しています。 このパーセンテージが高くなるに連れて、割り当ての効率がよくなります。 各論理ボリュームには独自のイントラポリシーがあります。 オペレーティング・システムは、この要件を満たせない場合、要件を満たすための最良の方法を選択します。本書の例では、合計 114 のロジカル・パーティション (LP) があり、42 の LP が中間に、26 の LP がセンターに、46 の LP が内部エッジにあります。 論理ボリュームのイントラポリシーはセンターなので、「in-band」は 22% (26 / (42+26+46)) になります。「DISTRIBUTION」は、物理パーティションがイントラポリシーの各部分 (下記) にどのように配置されているかを示しています。

```
edge : middle : center : inner-middle : inner-edge
```

物理パーティションの配置に関する追加情報は、 217 ページの『物理ボリューム上の位置』を参照してください。

lslv コマンドによるデータの物理的配置の評価

ワークロードがかなり高い入出力依存性を示している場合、ディスク上のファイルの配置を調べて、あるレベルで再編成を行えば改善が得られるかどうかを判断してください。

物理ボリューム hdisk0 内の論理ボリューム hd11 のパーティションの配置を見るには、以下を使用します。

```
# lslv -p hdisk0 hd11
hdisk0:hd11:/home/op
USED  USED  USED  USED  USED  USED  USED  USED  USED  USED     1-10
USED  USED  USED  USED  USED  USED  USED                      11-17

USED  USED  USED  USED  USED  USED  USED  USED  USED  USED    18-27
USED  USED  USED  USED  USED  USED  USED                      28-34

USED  USED  USED  USED  USED  USED  USED  USED  USED  USED    35-44
USED  USED  USED  USED  USED  USED                            45-50

USED  USED  USED  USED  USED  USED  USED  USED  USED  USED    51-60
0052  0053  0054  0055  0056  0057  0058                      61-67

0059  0060  0061  0062  0063  0064  0065  0066  0067  0068    68-77
0069  0070  0071  0072  0073  0074  0075                      78-84
```

hdisk1 上にある hd11 の残りの部分について見るには、次のようにします。

```
# lslv -p hdisk1 hd11
hdisk1:hd11:/home/op
0035  0036  0037  0038  0039  0040  0041  0042  0043  0044    1-10
0045  0046  0047  0048  0049  0050  0051                     11-17

USED  USED  USED  USED  USED  USED  USED  USED  USED  USED   18-27
USED  USED  USED  USED  USED  USED  USED                     28-34

USED  USED  USED  USED  USED  USED  USED  USED  USED  USED   35-44
USED  USED  USED  USED  USED  USED                           45-50

0001  0002  0003  0004  0005  0006  0007  0008  0009  0010   51-60
0011  0012  0013  0014  0015  0016  0017                     61-67

0018  0019  0020  0021  0022  0023  0024  0025  0026  0027   68-77
0028  0029  0030  0031  0032  0033  0034                     78-84
```

上から下へ向けて、5 つのブロックがそれぞれエッジ、ミドル、センター、内部ミドル、内部エッジを表しています。

- USED は、このロケーションにある物理パーティションが指定以外の論理ボリュームによって使用されることを示します。 数字は、**lslv -p** コマンドによって指定された論理ボリュームのロジカル・パーティション番号を示します。
- 「FREE」は、この物理パーティションがどの論理ボリュームによっても使用されないことを示します。ロジカル・パーティションがディスクをまたがって連続していない場合、論理ボリュームのフラグメント化が起こります。
- 「STALE」物理パーティションは、ユーザーが使用できないデータを含む物理パーティションです。「STALE」物理パーティションは、 **lspv -m** コマンドでも見ることができます。 「STALE」とマークされた物理パーティションは、有効な物理パーティションと同じ情報を含むように更新する必要があります。 このプロセスは、**syncvg** コマンドによる再同期と呼ばれ、varyon する時に実行できるか、またはシステムの実行中はいつでも始動できます。「STALE」パーティションは、有効なデータで再書き込みされるまでは、読み取り要求を満たすために使用されることも、書き込み要求時に書き込まれることもありません。

前の例では、論理ボリューム hd11 が物理ボリューム hdisk1 の内部で、その内部ミドルの最初のロジカル・パーティションと hdisk1 の内部領域によってフラグメント化され、一方ロジカル・パーティション 35 から 51 は外部領域にあります。 hd11 をランダムにアクセスしたワークロードでは、論理ボリューム hd11 でシークに時間がかかる可能性があるので、不必要な入出力待ち時間が発生することになります。 これらのレポートは、hdisk0 または hdisk1 のどちらにも、空き物理パーティションがないことを示しています。

fileplace コマンドによるファイル配置の評価

以前にコピーされたファイル (big1) がディスクにどのように保管されたかを見るには、**fileplace** コマンドを使用することができます。 **fileplace** コマンドでは、論理ボリューム内または 1 つ以上の物理ボリューム内のファイル・ブロックの配置を表示します。

fileplace コマンドがインストールされていて、使用可能かどうかを判別するには、次のコマンドを実行します。

```
# lslpp -lI perfagent.tools
```

次のコマンドを使用します。

```
# fileplace -pv big1

File: big1  Size: 3554273 bytes  Vol: /dev/hd10
Blk Size: 4096  Frag Size: 4096  Nfrags: 868   Compress: no
Inode: 19  Mode: -rwxr-xr-x  Owner: hoetzel  Group: system

  Physical Addresses (mirror copy 1)                          Logical Fragment
  ----------------------------------                          ----------------
  0001584-0001591  hdisk0     8 frags    32768 Bytes,   0.9%    0001040-0001047
  0001624-0001671  hdisk0    48 frags   196608 Bytes,   5.5%    0001080-0001127
  0001728-0002539  hdisk0   812 frags  3325952 Bytes,  93.5%    0001184-0001995

  868 frags over space of 956 frags:  space efficiency = 90.8%
  3 fragments out of 868 possible:  sequentiality = 99.8%
```

この例では、ファイル内に非常に小さいフラグメント化があり、それらが小さいギャップであることを示しています。 したがって、big1 のディスク配置は、その順次読み取り時間に重大な影響は与えていないことを推論できます。 さらに、(最近作成した) 3.5 MB のファイルがこの小さいフラグメントに出会うとすると、ファイルシステムは一般に特にフラグメント化にはなっていないように思われます。

ときおり、ファイルの部分がボリューム内のどのブロックにもマップされない場合があります。 このような領域は、ファイルシステムによって暗黙的にゼロが充てんされています。 この領域は、unallocated 論理ブロックとして表示されています。 このようなホールがあるファイルは、実際に占めているより大きいバイト数を示す場合があります (すなわち、**ls -l** コマンドは大きいサイズを示すのに対して、**du** コマンドは小さいサイズまたはそのファイルが実際にディスク上で占めているブロック数を示す)。

fileplace コマンドは、論理ボリュームからファイルのブロックのリストを読み取ります。 新しいファイルの場合、その情報がまだディスクに入っていない可能性があります。 **sync** コマンドを使用して、情報をフラッシュしてください。 また、**fileplace** コマンドの場合も、NFS リモート・ファイルが表示されません (コマンドがサーバーで実行されている場合を除く)。

注**:** さまざまなロケーションに対してシークし、広く分散したレコードを書き込むことによってファイルが作成された場合、レコードを含むページだけがディスク上でスペースを取り、**fileplace** レポートに表示されます。 ファイルシステムでは、ファイルを作成するときに、間にはさまっているページを自動的に充てんしません。ただし、そのようなファイルを順次に読み取る場合 (例えば、**cp** または **tar** コマンドによっ

て)、レコード間のスペースは 2 進ゼロとして読み取られます。 つまり、そのような **cp** コマンドの出力は、データが同じであっても、入力ファイルよりかなり大きくなる可能性があります。

スペース効率と順次性**:**

スペース効率が高いというのは、ファイルのフラグメント化が少なく、順次ファイルのアクセスにより適していることを示します。 順次性が高いというのは、ファイルがより連続して配置されていることであり、順次ファイルのアクセスにより適していることを示します。

スペース効率 **=**
: ファイル・ストレージに使用されたフラグメントの総数 / (最大フラグメント物理アドレス - 最小フラグメント物理アドレス + 1)

順次性 **=**
: (合計フラグメント数 - グループ化されたフラグメントの数 +1) / 合計フラグメント数

順次性またはスペース効率の値が低くなったことが分かったら、**reorgvg** コマンドを使用して、論理ボリュームの使用率と効率を改善することができます (224 ページの『論理ボリュームの再編成』を参照)。 ファイルシステムの使用率と効率を改善する場合は、 264 ページの『ファイルシステムの再編成』を参照してください。

この例では、最大フラグメント物理アドレス - 最小フラグメント物理アドレス + 1 は、0002539 - 0001584 + 1 = 956 フラグメント、合計使用フラグメントは、8 + 48 + 812 = 868、スペース効率は、868 / 956 (90.8%)、順次性は (868 - 3 + 1) / 868 = 99.8% です。

ファイル保管に使用される合計フラグメント数には、間接ブロックのロケーションは含まれないので、 32 KB より大きいファイルの場合、ファイルが連続するフラグメントに置かれている場合でも、スペース効率は 100% を超えることはありません。

vmstat コマンドによるページング・スペース入出力の評価

vmstat のレポートは、発生するページング・スペース入出力の量を表します。

ページ・スペースからの入出力はランダムで、ほとんどの場合、一度に 1 ページずつです。 次の例は両方とも、**rmss** コマンドを使用して人為的に縮めたマシンにおける、 C のコンパイル中に発生するページング・アクティビティーを示しています。 「*pi*」および「*po*」(ページング・スペースのページインおよびページング・スペースのページアウト) 欄は、各 5 秒間隔の間のページング・スペース入出力 (4096 バイト・ページ単位で) を示しています。 最初のレポート (システム・リブート以来の要約) は削除されています。 ページング・アクティビティーはバーストで発生することに注意してください。

```
# vmstat 5 8
kthr     memory             page              faults        cpu
----- ----------- ------------------------ ------------ -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 0  1 72379   434   0   0   0   0    2   0 376  192 478  9  3 87  1
 0  1 72379   391   0   8   0   0    0   0 631 2967 775 10  1 83  6
 0  1 72379   391   0   0   0   0    0   0 625 2672 790  5  3 92  0
 0  1 72379   175   0   7   0   0    0   0 721 3215 868  8  4 72 16
 2  1 71384   877   0  12  13  44  150   0 662 3049 853  7 12 40 41
 0  2 71929   127   0  35  30 182  666   0 709 2838 977 15 13  0 71
 0  1 71938   122   0   0   8  32  122   0 608 3332 787 10  4 75 11
 0  1 71938   122   0   0   0   3   12   0 611 2834 733  5  3 75 17
```

次に示す「before と after」の **vmstat -s** レポートは、ページング・アクティビティーの累積の様子を示しています。 本当のページング・スペース入出力を表しているのは、ページング・スペースのページインとページング・スペースのページアウトであることを忘れないでください。(不適当な) ページインおよび

ページアウトは、ページング・メカニズムによって実行された合計入出力、すなわち、ページング・スペース入出力と通常ファイル入出力の両方を報告します。 レポートでは、このディスカッションに無関係の行は削除するように編集されています。

# vmstat -s # before	**# vmstat -s # after**
6602 ページイン 3948 ページアウト **544** ページング・スペースからのページイン **1923** ページング・スペースからのページアウト 0 再利用の合計	7022 ページイン 4146 ページアウト **689** ページング・スペースからのページイン **2032** ページング・スペースからのページアウト 0 再利用の合計

コンパイル中にページング・スペースのページアウトより多くのページインが行われたという事実は、システムをスラッシングが始まる点まで縮小させたことを示唆しています。 一部のページは、フレームが使用の完了前にスチールされたために再ページされています。

vmstat コマンドによる総合的なディスク入出力の評価

ここまでに説明した手法は、プログラムによって生成されたディスク入出力負荷の評価にも使用できます。

システムがほかには何も実行していないとすると、次のシーケンス、

```
# vmstat -s >statout
# testpgm
# sync
# vmstat -s >> statout
# egrep "ins|outs" statout
```

では、次のようなディスク・アクティビティー累積カウントの前と後の状態を生成します。

```
     5698 page ins
     5012 page outs
        0 paging space page ins
       32 paging space page outs
     6671 page ins
     5268 page outs
        8 paging space page ins
      225 paging space page outs
```

このコマンド (大規模な C コンパイル) の実行中に、システムは合計 981 ページを読み取り (ページング・スペースから 8)、合計 449 ページを書き込みました (ページング・スペースへ 193)。

filemon コマンドによる詳細な入出力分析

filemon コマンドは、トレース機能を使用して、論理ファイルシステム、仮想メモリー・セグメント、LVM、および物理ディスク・レイヤーを含む、ファイルシステム使用率のさまざまなレイヤーでの、ある時間間隔中の入出力アクティビティーの詳細を取得します。

filemon コマンドは、すべてのレイヤーにあるデータを収集するために使用するか、または **-O** レイヤー・オプションでレイヤーを指定することができます。デフォルトは、VM、LVM、および物理層でのデータ収集です。 要約と詳細の両方のレポートが生成されます。 これはトレース機能を使用するので、**filemon** コマンドは root ユーザーまたはシステム・グループのメンバーしか実行できません。

filemon コマンドがインストールされていて、使用可能かどうかを判別するには、次のコマンドを実行します。

```
# lslpp -lI perfagent.tools
```

トレースは **filemon** コマンドによって開始され、オプションで **trcoff** サブコマンドによって中断され、**trcon** サブコマンドによって再開されます。トレースが終了するとすぐに、**filemon** コマンドが、そのレポートを stdout に書き込みます。

注**: -u** フラグを指定しなければ、**filemon** コマンドの開始後にオープンされたファイルのデータだけが収集されます。

filemon コマンドは、リアルタイムのトレース・プロセスからではなく、指定されたファイルから入出力トレース・データを読み取ることができます。 この場合は、**filemon** レポートに、システムおよびトレース・ファイルによって表される期間の、入出力アクティビティーの要約を示します。このオフライン処理方式は、リモート・マシンからトレース・ファイルをポストプロセスする必要があるとき、またはトレース・データの収集を実行してから、また別のときにポストプロセスする必要があるときに、役立ちます。

trcrpt -r コマンドは、次のように、トレース・ログ・ファイル上で実行し、別のファイルへリダイレクトする必要があります。

```
# gennames > gennames.out
# trcrpt -r   trace.out > trace.rpt
```

ここで、次のように、調整済みのトレース・ログ・ファイルが **filemon** コマンドに送られ、前に記録されたトレース・セッションによって取り込まれた入出力アクティビティーに関して報告が行われます。

```
# filemon -i trace.rpt -n gennames.out | pg
```

この例では、**filemon** コマンドは、trace.rpt 入力ファイルから、ファイルシステムのトレース・イベントを読み取ります。 トレース・データは既にファイルに取り込まれているので、**filemon** コマンドは自分自身をバックグラウンドにして、アプリケーション・プログラムが実行できるようにすることはありません。 ファイル全体が読み取られた後で、仮想メモリー、論理ボリューム、および物理ボリュームのレベルの入出力アクティビティー・レポートが標準出力に表示されます (この例では、**pg** コマンドにパイピングされます)。

-C all フラグが指定されて **trace** コマンドが実行された場合は、同様に **-C all** フラグを指定して **trcrpt** コマンドを実行してください (434 ページの『トレース -C 出力によるレポートのフォーマット設定』を参照してください)。

次のコマンドのシーケンスは、**filemon** コマンドの使用法の例を示します。

```
# filemon -o fm.out -O all; cp /smit.log /dev/null ; trcstop
```

ほかには何も実行していないシステムで、このシーケンスによって作成されるレポートは、次のようになります。

```
Thu Aug 19 11:30:49 1999
System: AIX texmex Node: 4 Machine: 000691854C00

0.369 secs in measured interval
Cpu utilization:  9.0%

Most Active Files
------------------------------------------------------------------------
  #MBs  #opns   #rds   #wrs  file                 volume:inode
------------------------------------------------------------------------
   0.1      1     14      0  smit.log             /dev/hd4:858
   0.0      1      0     13  null
   0.0      2      4      0  ksh.cat              /dev/hd2:16872
   0.0      1      2      0  cmdtrace.cat         /dev/hd2:16739

Most Active Segments
------------------------------------------------------------------------
  #MBs  #rpgs  #wpgs  segid  segtype              volume:inode
------------------------------------------------------------------------
   0.1     13      0   5e93  ???
   0.0      2      0   22ed  ???
   0.0      1      0   5c77  persistent
```

```
Most Active Logical Volumes
------------------------------------------------------------------------
  util  #rblk  #wblk   KB/s  volume               description
------------------------------------------------------------------------
  0.06    112      0  151.9  /dev/hd4             /
  0.04     16      0   21.7  /dev/hd2             /usr

Most Active Physical Volumes
------------------------------------------------------------------------
  util  #rblk  #wblk   KB/s  volume               description
------------------------------------------------------------------------
  0.10    128      0  173.6  /dev/hdisk0          N/A




------------------------------------------------------------------------
Detailed File Stats
------------------------------------------------------------------------

FILE: /smit.log  volume: /dev/hd4 (/)  inode: 858
opens:                  1
total bytes xfrd:       57344
reads:                  14      (0 errs)
  read sizes (bytes):   avg  4096.0 min    4096 max    4096 sdev     0.0
  read times (msec):    avg   1.709 min   0.002 max  19.996 sdev   5.092

FILE: /dev/null
opens:                  1
total bytes xfrd:       50600
writes:                 13      (0 errs)
  write sizes (bytes):  avg  3892.3 min    1448 max    4096 sdev   705.6
  write times (msec):   avg   0.007 min   0.003 max   0.022 sdev   0.006

FILE: /usr/lib/nls/msg/en_US/ksh.cat  volume: /dev/hd2 (/usr)  inode: 16872
opens:                  2
total bytes xfrd:       16384
reads:                  4       (0 errs)
  read sizes (bytes):   avg  4096.0 min    4096 max    4096 sdev     0.0
  read times (msec):    avg   0.042 min   0.015 max   0.070 sdev   0.025
lseeks:                 10

FILE: /usr/lib/nls/msg/en_US/cmdtrace.cat  volume: /dev/hd2 (/usr)  inode: 16739
opens:                  1
total bytes xfrd:       8192
reads:                  2       (0 errs)
  read sizes (bytes):   avg  4096.0 min    4096 max    4096 sdev     0.0
  read times (msec):    avg   0.062 min   0.049 max   0.075 sdev   0.013
lseeks:                 8

------------------------------------------------------------------------
Detailed VM Segment Stats   (4096 byte pages)
------------------------------------------------------------------------

SEGMENT: 5e93  segtype: ???
segment flags:
reads:                  13      (0 errs)
  read times (msec):    avg   1.979 min   0.957 max   5.970 sdev   1.310
  read sequences:       1
  read seq. lengths:    avg    13.0 min      13 max      13 sdev     0.0

SEGMENT: 22ed  segtype: ???
segment flags:          inode
reads:                  2       (0 errs)
  read times (msec):    avg   8.102 min   7.786 max   8.418 sdev   0.316
  read sequences:       2
```

```
  read seq. lengths:    avg     1.0 min       1 max       1 sdev     0.0

SEGMENT: 5c77  segtype: persistent
segment flags:          pers defer
reads:                  1       (0 errs)
  read times (msec):    avg  13.810 min  13.810 max  13.810 sdev   0.000
  read sequences:       1
  read seq. lengths:    avg     1.0 min       1 max       1 sdev     0.0

------------------------------------------------------------------------
Detailed Logical Volume Stats   (512 byte blocks)
------------------------------------------------------------------------

VOLUME: /dev/hd4  description: /
reads:                  5       (0 errs)
  read sizes (blks):    avg    22.4 min       8 max      40 sdev    12.8
  read times (msec):    avg   4.847 min   0.938 max  13.792 sdev   4.819
  read sequences:       3
  read seq. lengths:    avg    37.3 min       8 max      64 sdev    22.9
seeks:                  3       (60.0%)
  seek dist (blks):     init   6344,
                        avg    40.0 min       8 max      72 sdev    32.0
time to next req(msec): avg  70.473 min   0.224 max 331.020 sdev 130.364
throughput:             151.9 KB/sec
utilization:            0.06

VOLUME: /dev/hd2  description: /usr
reads:                  2       (0 errs)
  read sizes (blks):    avg     8.0 min       8 max       8 sdev     0.0
  read times (msec):    avg   8.078 min   7.769 max   8.387 sdev   0.309
  read sequences:       2
  read seq. lengths:    avg     8.0 min       8 max       8 sdev     0.0
seeks:                  2       (100.0%)
  seek dist (blks):     init 608672,
                        avg    16.0 min      16 max      16 sdev     0.0
time to next req(msec): avg 162.160 min   8.497 max 315.823 sdev 153.663
throughput:             21.7 KB/sec
utilization:            0.04

------------------------------------------------------------------------
Detailed Physical Volume Stats   (512 byte blocks)
------------------------------------------------------------------------

VOLUME: /dev/hdisk0  description: N/A
reads:                  7       (0 errs)
  read sizes (blks):    avg    18.3 min       8 max      40 sdev    12.6
  read times (msec):    avg   5.723 min   0.905 max  20.448 sdev   6.567
  read sequences:       5
  read seq. lengths:    avg    25.6 min       8 max      64 sdev    22.9
seeks:                  5       (71.4%)
  seek dist (blks):     init 4233888,
                        avg 171086.0 min       8 max  684248 sdev 296274.2
  seek dist (%tot blks):init 48.03665,
                        avg 1.94110 min 0.00009 max 7.76331 sdev 3.36145
time to next req(msec): avg  50.340 min   0.226 max 315.865 sdev 108.483
throughput:             173.6 KB/sec
utilization:            0.10
```

filemon コマンドを実際のワークロードが実行されているシステムで使用すると、これよりかなり大きいレポートができるので、さらにトレース・バッファー・スペースが必要な場合があります。 **filemon** コマンドが消費するスペースおよび CPU 時間によって、システム・パフォーマンスがある程度低下する可能性があります。 実稼働環境で使用する前に、実動システム以外のシステムを使用して、**filemon** コマンドの実験を行ってください。 また、オフライン処理を使用し、多数の CPU があるシステムでは、**trace** コマンドで **-C all** フラグを使用してください。

注**: filemon** コマンドでは、詳細統計情報セクションに平均、最小、最大、および標準偏差が報告されますが、この結果を信頼区間の作成あるいはその他の正式な統計情報の推論に使用するべきではありません。一般に、データ・ポイントの分散はランダムでも対称でもありません。

filemon コマンドのグローバル・レポート**:**

グローバル・レポートには、測定された間隔中の最もアクティブなファイル、セグメント、論理ボリューム、および物理ボリュームがリストされます。

これらは、**filemon** レポートの先頭に表示されます。デフォルトでは、論理ファイルと仮想メモリーのレポートは、それぞれ転送されたデータの合計量に基づいて、最もアクティブな 20 個のファイルとセグメントに限定されています。 **-v** フラグが指定されている場合、すべてのファイルおよびセグメントのアクティビティーが報告されます。 レポート内の情報は、最もアクティブなものから順に、上から下へリストされます。

最もアクティブなファイル*:*

filemon コマンドは、論理ファイルシステム、仮想メモリー・セグメント、LVM、および物理ディスク・レイヤーを含む、ファイルシステム使用率のさまざまなレイヤーでの最もアクティブなファイルをリストするレポートの作成に使用できます。 このセクションはレポートに表示される列見出しの説明です。

#MBs 測定間隔中にこのファイルに関して転送された合計 MB 数。各行は、このフィールドによって降順にソートされています。

#opns 測定期間中のファイルのオープン回数。

#rds ファイルに対する読み取り呼び出しの数。

#wrs ファイルに対する書き込み呼び出しの数。

file ファイル名 (絶対パス名は詳細レポートに示される)。

volume:inode
ファイルがある論理ボリュームと関連ファイルシステム内でのファイルの i ノード番号。 このフィールドは、ファイルを、詳細 VM セグメント・レポートに示される対応する永続セグメントと関連付けるために使用することができます。 このフィールドは、実行時に作成、削除される一時ファイルでは、ブランクの場合があります。

最もアクティブなファイルは、論理ボリューム hd4 の `smit.log` と **null** ファイルです。 アプリケーションは、terminfo データベースを画面管理に利用するので、`ksh.cat` および `cmdtrace.cat` も使用中です。画面にメッセージを追加する必要がある場合、シェルは、いつでもデータのソースのカタログを使用します。

不明ファイルを識別するには、論理ボリューム名 (`/dev/hd1`) をファイルシステムのマウント・ポイントである `/home` に変換して、**find** または **ncheck** コマンドを使用することができます。

```
# find / -inum 858 -print
/smit.log
```

または

```
# ncheck -i 858 /
/:
858     /smit.log
```

最もアクティブなセグメント*:*

filemon コマンドは、論理ファイルシステム、仮想メモリー・セグメント、LVM、および物理ディスク・レイヤーを含む、ファイルシステム使用率のさまざまなレイヤーでの最もアクティブなセグメントをリストするレポートの作成に使用できます。 このセクションはレポートに表示される列見出しの説明です。

#MBs 測定間隔中にこのセグメントに関して転送された合計 MB 数。各行は、このフィールドによって降順にソートされています。

#rpgs ディスクからセグメントへ読み込まれた 4 KB ページの数。

#wpgs
セグメントからディスクへ書き出された (ページアウト) 4 KB ページの数。

#segid
メモリー・セグメントの VMM ID。

segtype
セグメントの型: 作業セグメント、永続セグメント (ローカル・ファイル)、クライアント・セグメント (リモート・ファイル)、ページ・テーブル・セグメント、システム・セグメント、またはファイルシステム・データ (ログ、ルート・ディレクトリー、
.inode、.inodemap、.inodex、.inodexmap、.indirect、.diskmap) を含む特殊永続セグメント。

volume:inode
永続セグメントの場合、関連するファイルとファイルの i ノード番号を含む論理ボリュームの名前。 このフィールドを使用すると、永続セグメントを、Detailed File Stats レポートに示される対応するファイルと関連付けることができます。このフィールドは、永続セグメント以外のセグメントの場合はブランクです。

コマンドがまだアクティブの場合、**svmon -D** *segid* のように、セグメント ID (segid) を指定すれば、仮想メモリー分析ツール **svmon** を使用して、セグメントに関する詳細情報を表示できます。 詳しい説明は、『svmon コマンド』を参照してください。

本書の例では、segtype の **???** は、システムがセグメント・タイプを識別できないので、 **svmon** コマンドを使用して詳細情報を得る必要があることを意味しています。

最もアクティブな論理ボリューム*:*

filemon コマンドは、論理ファイルシステム、仮想メモリー・セグメント、LVM、および物理ディスク・レイヤーを含む、ファイルシステム使用率のさまざまなレイヤーでの最もアクティブな論理ボリュームをリストするレポートの作成に使用できます。 このセクションはレポートに表示される列見出しの説明です。

util 論理ボリュームの使用率。

#rblk 論理ボリュームから読み取った 512 バイト・ブロックの数。

#wblk
論理ボリュームへ書き込まれた 512 バイト・ブロックの数。

KB/s 平均データ転送速度 (1 秒当たり KB)。

volume
論理ボリューム名。

description
ファイルシステムのマウント・ポイントまたは論理ボリューム・タイプ (paging、jfslog、boot、または sysdump)。 例えば、論理ボリューム `/dev/hd2` は `/usr` で、`/dev/hd6` は paging、そして

/dev/hd8 は jfslog です。 *compressed* という語が使用される場合もあります。 これは、すべてのデータがディスクに書き出される前に、Lempel-Zev (LZ) 圧縮を使用して自動的に圧縮され、またすべてのデータがディスクから読み込まれるときに自動的に圧縮解除されることを意味します (詳細については、 263 ページの『JFS の圧縮』を参照)。

使用率はパーセンテージで表され、0.06 は測定間隔中に 6% 使用中であったことを示しています。

最もアクティブな物理ボリューム*:*

filemon コマンドは、論理ファイルシステム、仮想メモリー・セグメント、LVM、および物理ディスク・レイヤーを含む、ファイルシステム使用率のさまざまなレイヤーでの最もアクティブな物理ボリュームをリストするレポートの作成に使用できます。 このセクションはレポートに表示される列見出しの説明です。

util 物理ボリュームの使用率。

注**:** 論理ボリュームの入出力要求は、物理ボリュームの入出力要求の前に始まり、後に終了します。 したがって、論理ボリュームの合計使用率は物理ボリュームの合計使用率より高く表示されます。

#rblk 物理ボリュームから読み取られた 512 バイト・ブロックの数。

#wblk
物理ボリュームへ書き込まれた 512 バイト・ブロックの数。

KB/s 平均データ転送速度 (1 秒当たり KB)。

volume
物理ボリューム名。

description
物理ボリューム・タイプの簡単な説明。例えば、SCSI マルチメディア CD-ROM ドライブや 16 ビット SCSI ディスク・ドライブなど。

使用率はパーセンテージで表され、0.10 は測定間隔中に 10% 使用中であったことを示しています。

最もアクティブなファイル *(*プロセス順にソート*):*

filemon コマンドは、論理ファイルシステム、仮想メモリー・セグメント、LVM、およびプロセス順にソートされた物理ディスク・レイヤーを含む、ファイルシステム使用のさまざまなレイヤーでの、最もアクティブなファイルをリストするレポートの作成に使用できます。

#MBS ファイルとのやりとりで転送されるメガバイトの累計。各行は、このフィールドによって、降順にソートされています。

#opns 測定中にファイルがオープンされた回数。

#rds ファイルに対するシステム呼び出しの読み取りの数。

#wrs ファイルに対するシステム呼び出しの書き込みの数。

file ファイルの名前。絶対パス名は詳細レポートにあります。

PID ファイルをオープンしたプロセスの ID。

プロセス
ファイルをオープンしたプロセスの名前。

TID ファイルをオープンしたスレッドの ID。

最もアクティブなファイル *(*スレッド順にソート*):*

filemon コマンドは、論理ファイルシステム、仮想メモリー・セグメント、LVM、およびスレッド順にソートされた物理ディスク・レイヤーを含む、ファイルシステム使用のさまざまなレイヤーでの、最もアクティブなファイルをリストするレポートの作成に使用できます。

#MBS ファイルとのやりとりで転送されるメガバイトの累計。各行は、このフィールドによって、降順にソートされています。

#opns 測定期間中にファイルがオープンされた回数。

#rds ファイルに対するシステム呼び出しの読み取りの数。

#wrs ファイルに対するシステム呼び出しの書き込みの数。

file ファイルの名前。絶対パス名は詳細レポートにあります。

PID ファイルをオープンしたプロセスの ID。

プロセス
ファイルをオープンしたプロセスの名前。

TID ファイルをオープンしたスレッドの ID。

filemon コマンドの詳細レポート**:**

詳細レポートには、グローバル・レポートの追加情報が表示されます。

詳細レポートで報告されるファイル、セグメント、またはボリュームごとに 1 つのエントリーがあります。 各エントリーのフィールドについては、4 つの詳細レポートについて以下で説明します。 一部のフィールドでは 1 つの値を報告し、他のフィールドでは多くの値の分散の特徴を示す統計情報を報告します。例えば、応答時間統計情報はモニターしたすべての読み取りおよび書き込み要求について記録されます。平均、最小、および最大の応答時間だけではなく、応答時間の標準偏差も報告されます。 標準偏差は、個々の応答時間が平均値からどれだけ逸脱しているかを示すために使用されます。サンプリングした応答時間の約 2/3 が、平均値 - 標準偏差 (avg - sdev) と平均値 + 標準偏差 (avg + sdev) の間に入ります。 応答時間の分散が広範囲に散らばっている場合、標準偏差は平均応答時間に比べて大きくなります。

詳細ファイル統計情報*:*

詳細ファイル統計情報は、「*Most Active Files*」レポートにリストされている各ファイルに関して提供されます。

「*Most Active Files*」レポートのスタンザを使用すると、ファイルに対してどのアクセスが行われたかを判断することができます。 転送、オープン、読み取り、書き込み、および lseek の合計バイト数に加えて、読み取り/書き込みのサイズと回数も判別できます。

FILE ファイルの名前。 できれば、絶対パス名を指定します。

volume
ファイルを含む論理ボリューム/ファイルシステムの名前。

inode ファイルシステム内のファイルの i ノード番号。

opens モニター中にファイルがオープンされた回数。

total bytes xfrd
ファイルからの読み取り/ファイルへの書き込みの合計バイト数。

reads ファイルに対する読み取り呼び出しの数。

read sizes (bytes)
読み取り転送サイズ統計情報 (avg/min/max/sdev)、バイト単位。

read times (msec)
読み取り応答時間統計情報 (avg/min/max/sdev)、ミリ秒単位。

writes
ファイルに対する書き込み呼び出しの数。

write sizes (bytes)
書き込み転送サイズ統計情報。

write times (msec)
書き込み応答時間統計情報。

lseeks
lseek() サブルーチン呼び出しの数。

読み取りサイズと書き込みサイズから、アプリケーションが情報の読み取りと書き込みを行う際の効率が分かります。 4 KB ページの倍数を使用すると、最良の結果が得られます。

VM セグメントの詳細な統計情報**:**

「Most Active Segments」レポートは、すべての VM セグメントの詳細な統計情報をリストします。

「Most Active Segments」レポートにリストされているエレメントには、それぞれ対応するスタンザがあり、メモリーに関する実際の入出力の詳細情報が表示されています。

SEGMENT
オペレーティング・システム内部のセグメント ID。

segtype
セグメントの内容のタイプ。

segment flags
各種のセグメント属性。

volume
永続セグメントの場合、対応するファイルを含む論理ボリュームの名前。

inode 永続セグメントの場合、対応するファイルの i ノード番号。

reads セグメントに読み込まれた (すなわち、ページインされた) 4096 バイト・ページの数。

read times (msec)
読み取り応答時間統計情報 (avg/min/max/sdev)、ミリ秒単位。

read sequences
読み取りシーケンスの数。 シーケンスとは、連続して読み込まれた (ページインされた) 一連のページのことです。 読み取りシーケンスの数は、順次アクセスの量を示すインディケーターです。

read seq. lengths
読み取りシーケンスの長さを記述する統計情報、ページ単位。

writes
セグメントからディスクに書き出された (すなわち、ページアウトされた) ページの数。

write times (msec)
書き込み応答時間統計情報。

write sequences
書き込みシーケンスの数。 シーケンスとは、連続して書き込まれた (ページアウトされた) 一連のページのことです。

write seq. lengths
書き込みシーケンスの長さを記述する統計情報、ページ単位。

読み取りシーケンスと書き込みシーケンスのカウントを調べることによって、アクセスが順次かランダムかを判別することができます。 例えば、読み取りシーケンスのカウントが読み取りカウントに達した場合、ファイル・アクセスはランダムであることを示します。 それに対して、読み取りシーケンスのカウントが読み取りカウントより非常に小さい場合、読み取りシーケンス長の値が高ければ、ファイル・アクセスは順次であることを示します。書き込みと書き込みシーケンスについても、同じことが言えます。

論理ボリュームまたは物理ボリュームの詳細な統計情報*:*

「Most Active Logical Volumes / Most Active Physical Volumes」レポートにリストされているエレメントには、それぞれ対応するスタンザがあり、論理/物理ボリュームに関する詳細情報が表示されています。

読み取りと書き込みの回数のほかに、ユーザーは、論理ボリュームまたは物理ボリュームの初期シーク距離と平均シーク距離ばかりでなく、読み取りと書き込みの回数とサイズも判別することができます。

VOLUME
ボリュームの名前。

description
ボリュームの記述。 (論理ボリュームを扱う場合は、内容を記述し、物理ボリュームを扱う場合は、タイプを記述する。)

reads ボリュームに対して行われた読み取り要求の数。

read sizes (blks)
読み取り転送サイズ統計情報 (avg/min/max/sdev)、512 バイト・ブロック単位。

read times (msec)
読み取り応答時間統計情報 (avg/min/max/sdev)、ミリ秒単位。

read sequences
読み取りシーケンスの数。 シーケンスとは、連続して読み取られた一連の 512 バイト・ブロックのことです。 これは、順次アクセスの量を示します。

read seq. lengths
読み取りシーケンスの長さ (ブロック単位) を記述する統計情報。

writes
ボリュームに対して行われた書き込み要求の数。

write sizes (blks)
書き込み転送サイズ統計情報。

write times (msec)
書き込み応答時間統計情報。

write sequences
書き込みシーケンスの数。 シーケンスとは、連続して書き込まれた一連の 512 バイト・ブロックのことです。

write seq. lengths
書き込みシーケンスの長さ (ブロック単位) を記述する統計情報。

seeks 読み取りまたは書き込み要求の先にあったシークの数。シークを必要とした読み取りおよび書き込みの合計のパーセンテージの形でも表されます。

seek dist (blks)
シーク距離統計情報 (512 バイト・ブロック単位)。 通常の統計情報 (avg/min/max/sdev) に加えて、初期シーク操作の距離 (ブロック 0 が開始位置であるとして) は別に報告されます。 このシーク距離は、時には非常に大きくなります。これは、他の統計情報をゆがめるのを避けるために、別に報告されます。

seek dist (cyls)
(物理ボリュームのみ) ディスク・シリンダー単位のシーク距離統計情報。

time to next req
ボリュームに対する連続した読み取りまたはまたは書き込みの間の、時間の長さ (ミリ秒単位) を記述する統計情報 (avg/min/max/sdev)。 この欄は、ボリュームがアクセスされる速度を示します。

throughput
合計ボリューム・スループット (1 秒当たり KB 単位)。

utilization
ボリュームが使用中だった時間の部分。 このレポートのエントリーは、このフィールドによって降順にソートされています。

シーク時間が長いと、入出力応答時間が長くなり、その結果アプリケーションのパフォーマンスが低下します。 読み取りシーケンスと書き込みシーケンスのカウントを調べることによって、アクセスが順次かランダムかを判別することができます。 書き込みと書き込みシーケンスについても、同じロジックが当てはまります。

詳細ファイル統計情報 *(*プロセス順にソート*):*

詳細ファイル統計情報は、プロセス順にソートされた「Most Active Files」レポートにリストされている各ファイルに関して提供されます。

Process Id
ファイルをオープンしたプロセスの ID。

Name オープンしたファイルの名前 (パスを含む)。

Thread Id
ファイルをオープンしたスレッドの ID。

# of seeks
シークの数。

# of reads
読み取り操作の回数。

read errors
読み取りエラーの回数。

# of writes
書き込み操作の回数。

write errors
書き込みエラーの回数。

Bytes Read
読み取られたバイト数。

min 同時に読み取られたバイトの最小数。

avr 同時に読み取られたバイトの平均の数。

max 同時に読み取られたバイトの最大数。

Bytes Written
読み取られたバイト数。

min 同時に書き込まれたバイトの最小数。

avr 同時に書き込まれたバイトの平均の数。

max 同時に書き込まれたバイトの最大数。

Read Time
読み取り操作で消費された時間

Write Time
書き込み操作で消費された時間

詳細ファイル統計情報 *(*スレッド順にソート*):*

詳細ファイル統計情報は、スレッド順にソートされた「Most Active Files」レポートにリストされている各ファイルに関して提供されます。

Thread Id
ファイルをオープンしたスレッドの ID。

Name オープンしたファイルの名前 (パスを含む)。

Process Id
ファイルをオープンしたプロセスの ID。

# of seeks
シークの数。

# of reads
読み取り操作の回数。

read errors
読み取りエラーの回数。

# of writes
書き込み操作の回数。

write errors
書き込みエラーの回数。

Bytes Read
読み取られたバイト数。

min 同時に読み取られたバイトの最小数。

avr 同時に読み取られたバイトの平均の数。

max 同時に読み取られたバイトの最大数。

Bytes Written
読み取られたバイト数。

min 同時に書き込まれたバイトの最小数。

avr 同時に書き込まれたバイトの平均の数。

max 同時に書き込まれたバイトの最大数。

Read Time
読み取り操作で消費された時間

Write Time
書き込み操作で消費された時間

filemon コマンドの使用のガイドライン:

filemon コマンドの使用に関するガイドラインをいくつか示します。

- /etc/inittab ファイルは常に非常にアクティブです。 /etc/inittab で指定されたデーモンは、再作成の必要があるかどうかの判別のために、定期的にチェックされます。
- /etc/passwd ファイルも、常に非常にアクティブです。ファイルとディレクトリーのアクセス権限がチェックされます。
- シーク時間が長いと、入出力応答時間が長くなり、パフォーマンスが低下します。
- 読み取りと書き込みの大部分にシークが必要な場合、同じ物理ディスク上に、フラグメント化されたファイルや過度にアクティブなファイルシステムがある可能性があります。ただし、オンライン・トランザクション処理 (OLTP) またはデータベース・システムの場合、この動作は正常です。
- 読み取りと書き込みの数がシーケンスの数に達すると、物理ディスクのアクセスは順次よりランダムになります。 シーケンスとは、連続して読み取り (ページイン) または書き込み (ページアウト) された一連のページのことです。 seq. lengths はシーケンスの長さで、ページ単位です。 ランダム・ファイル・アクセスでは、多数のシークが行われる場合があります。 この場合、ファイル・アクセスがランダムなのか、ファイルがフラグメント化されているのかは、 **filemon** 出力からは区別できません。 さらに調べる場合は、**fileplace** コマンドを使用してください。
- リモート・ファイルは、「volume:inode」欄にリモート・システム名と一緒にリストされます。

filemon コマンドはある程度の CPU 能力を消費する可能性があるので、このツールは慎重に使用し、またシステム・パフォーマンスの分析にあたっては、このツールの実行によるオーバーヘッドを考慮に入れてください。テストから、CPU 飽和環境については以下のことが分かります。

- 入出力が少ない場合、**filemon** コマンドによって、大規模なコンパイルが約 1% 遅くなりました。
- ディスク出力の比率が高い場合、**filemon** コマンドによって、書き込みプログラムが約 5% 遅くなりました。

ディスク入出力のモニターの要約

一般に、「% iowait」が高いのは、システムにアプリケーション問題、メモリー不足、または非効率な入出力サブシステム構成などがあることを示しています。 例えば、アプリケーション問題は、多くの入出力を要求しながら、そのデータであまり多くの処理をしないことなどが原因として考えられます。 入出力のボトルネックを理解して、入出力サブシステムの効率を改善することが、このボトルネックの解決のかぎです。

ディスクに関する問題はさまざまで形で現れ、解決策もそれぞれ異なります。一般的な解決策としては次のものがあります。

- 特定の物理ディスクに置く、アクティブな論理ボリュームとファイルシステムの数を制限する。 このアイデアは、すべての物理ディスク・ドライブのファイル入出力を平均化することにあります。
- 論理ボリュームを複数の物理ディスクに分散して配置する。 これは、多数の異なるファイルをアクセスするときに特に役に立ちます。
- 複数のジャーナル・ファイルシステム (JFS) のログを 1 つのボリューム・グループに作成して、それらを個々のファイルシステムに割り当てる (可能なら、高速書き込みキャッシュ・デバイスの)。 これは、多数のファイル、特に一時ファイルを作成、削除、または変更するアプリケーションに便利です。
- **iostat** の出力が、ワークロードの入出力アクティビティーがシステム・ディスク・ドライブに一様に分散されていないことを示している場合や、同時に 1 つ以上のディスク・ドライブの使用率が 70% から 80% になることがよくある場合、フラグメント化を低減するためのファイルシステムのバックアップや復元など、ファイルの再編成を考える必要があります。フラグメント化が生じると、ドライブが過度にシークを行うようになり、応答時間全体の大部分を占めることになります。
- 大規模の、入出力集中のバックグラウンド・ジョブが対話式ジョブの応答時間の邪魔をしている場合は、入出力ペーシングを活動化することができます。
- 少数のファイルが繰り返し何度も読み取られているように見える場合、実メモリーを追加すれば、それらのファイルをより有効にバッファーに入れることができるかどうか、考えてみてください。
- ワークロードのアクセス・パターンが主にランダムの場合、ディスクの追加とランダムにアクセスされるファイルの複数ドライブへの分散を考慮します。
- ワークロードのアクセス・パターンが主に順次アクセスで、複数のディスク・ドライブを使用する場合は、1 つ以上のディスク・アダプターの追加を考慮します。また、ストライプ論理ボリュームを作成し、そこに大規模でパフォーマンスが重要な順次ファイルを入れることを考慮するのも、適切かもしれません。
- 高速キャッシュ・デバイスの使用。
- 非同期通信 I/O の使用。

lvmstat コマンドを使用した LVM パフォーマンス・モニター

lvmstat コマンドを使用して、論理ボリュームの特定の領域やパーティションへのアクセス頻度が他の領域やパーティションよりも高いかどうかを判別することができます。

これらアクセス頻度の高い領域の統計を **lvmstat** コマンドを使用して表示するには、論理ボリュームまたはボリューム・グループ単位で統計を実行できるようにする必要があります。

特定の論理ボリュームに対して **lvmstat** コマンドの統計を使用可能にするには、次のコマンドを使用します。

```
# lvmstat -l lvname -e
```

特定の論理ボリュームに対して **lvmstat** コマンドの統計を使用不可にするには、次のコマンドを使用します。

```
# lvmstat -l lvname -d
```

特定のボリューム・グループ内のすべての論理ボリュームに対して **lvmstat** コマンドの統計を使用可能にするには、次のコマンドを使用します。

```
# lvmstat -v vgname -e
```

特定のボリューム・グループ内のすべての論理ボリュームに対して **lvmstat** コマンドの統計を使用不可にするには、次のコマンドを使用します。

```
# lvmstat -v vgname -d
```

間隔値を指定せずに **lvmstat** コマンドを使用すると、出力には論理ボリューム内のすべてのパーティションの統計が表示されます。 間隔値を秒で指定すると、**lvmstat** コマンドは指定した間隔の間にアクセスされた特定のパーティションの統計のみを表示します。 以下に、**lvmstat** コマンドの例を示します。

```
# lvmstat -l lv00 1

Log_part  mirror#  iocnt   Kb_read   Kb_wrtn      Kbps
       1       1   65536     32768         0      0.02
       2       1   53718     26859         0      0.01

Log_part  mirror#  iocnt   Kb_read   Kb_wrtn      Kbps
       2       1    5420      2710         0  14263.16

Log_part  mirror#  iocnt   Kb_read   Kb_wrtn      Kbps
       2       1    5419      2709         0  15052.78

Log_part  mirror#  iocnt   Kb_read   Kb_wrtn      Kbps
       3       1    4449      2224         0  13903.12
       2       1     979       489         0   3059.38

Log_part  mirror#  iocnt   Kb_read   Kb_wrtn      Kbps
       3       1    5424      2712         0  12914
```

-c フラグを使用すると、**lvmstat** コマンドが表示する統計の数を制限できます。 **-c** フラグには、I/O アクティビティーが最も多いパーティションを表示する数を指定します。 以下は、**lvmstat** コマンドに **-c** フラグを指定して使用した例です。

```
# lvmstat -l lv00 -c 5
```

このコマンドを実行すると、I/O アクティビティーが最も多いパーティション 5 つの統計が表示されます。

反復パラメーターを指定しないと、**lvmstat** コマンドはコマンドを中断するまで出力を生成し続けます。逆にパラメーターを指定すると、**lvmstat** コマンドは指定した反復回数分の統計を表示します。

lvmstat コマンドを実際に使用してみて、ごく少数のパーティションの使用頻度だけが高いことに気付くことがありますが、その場合に **lvmstat** コマンドを使用して、それらのパーティションをいくつかのハード・ディスクに分散することもできます。 **lvmstat** コマンドを使用すると、元のハード・ディスクから別のハード・ディスクへパーティション単位でマイグレーションできます。 **lvmstat** コマンドの使用法について詳しくは、*Commands Reference, Volume 3*の『migratelp コマンド』を参照してください。

lvmstat コマンドについてオプションなどの詳しい情報は、 *Commands Reference, Volume 3*の『lvmstat コマンド』を参照してください。

パフォーマンスに影響する論理ボリューム属性

パフォーマンスに影響があり、論理ボリュームの作成時に制御できるさまざまな要因があります。 これらのオプションは、「**smitty mklv**」画面上に値の指定を求めるプロンプトとして表示されます。

物理ボリューム上の位置

*Intra-Physical Volume Allocation Policy (*物理ボリューム内割り振りポリシー*)* は、物理ボリューム上の物理パーティションを選択する際に、どのストラテジーを使用するかを指定します。 一般的な 5 つのストラテジーは、エッジ、内部エッジ、ミドル、内部ミドル、およびセンターです。

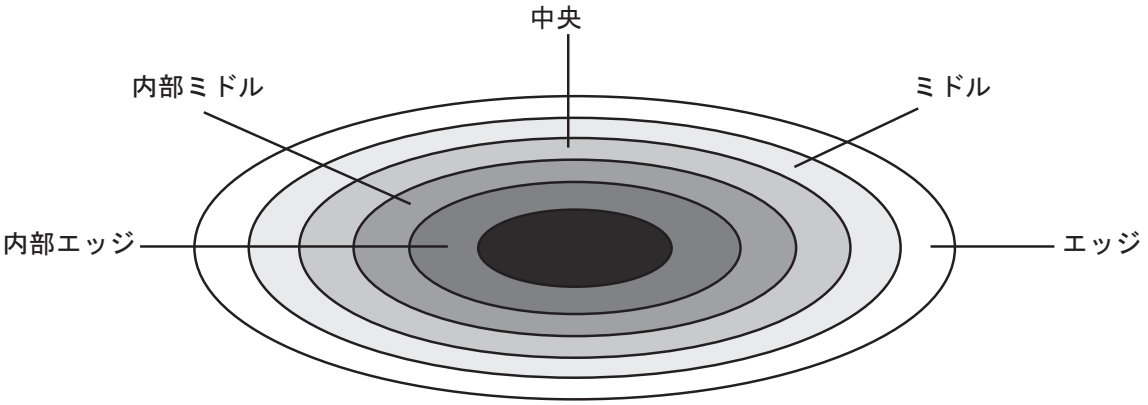


図 *17.* 物理ボリューム内割り振りポリシー： この図は、物理ボリュームまたはディスク上の記憶位置を示しています。ディスクは、ディスクの外部エッジからディスクの中心に向かって、数百のトラックにフォーマットされています。 ディスクの読み取り方法 (移動可能な読み取り/書き込みヘッドの下をトラックが回転する) のため、ディスクの中心近くに書き込まれたデータのシーク時間は、外部エッジに書き込まれたデータよりも速くなります。 この違いの主な理由は、読み取り/書き込みヘッドが機械的に動作することと、各トラックのセクターがヘッドの下を通過しなければならないことです。 データが中心に近づくにつれて、データの密度が高くなっていくので、ヘッドの物理的動作は少なくなります。 この結果、全体のスループットが高くなります。

物理パーティションは、1 番から始まり、一番外側のエッジから一番内側のエッジまで、連続して番号が付けられています。

エッジおよび内部エッジのストラテジーでは、物理ボリュームのエッジへのパーティションの割り当てを示します。 このようなパーティションは、平均シーク時間が最も遅く、どのようなアプリケーションで使用する場合も、一般に応答時間が長くなります。 1990 年代半ば以降に製造されたディスクでは、エッジの方が 1 トラック当たり保持できるセクターが多いので、順次入出力ではエッジの方が早くなります。

ミドルおよび内部ミドルのストラテジーでは、パーティションの割り当ての際に、物理ボリュームのエッジを避け、センターの範囲外にするように指定します。これらのストラテジーでは、パーティションに適度によい位置が割り当てられ、平均シーク時間は適度に良くなります。 物理ボリュームのパーティションの大部分が、このストラテジーを使用した割り当てに使用できます。

センター・ストラテジーでは、各物理ボリュームのセンター部分にパーティションを割り当てるように指定します。 このようなパーティションでは、平均シーク時間が最も早く、どのようなアプリケーションで使用する場合も、一般に応答時間が最良になります。 物理ボリューム上のパーティションが少ない場合は、他のどの一般的ストラテジーより、センター・ストラテジーに適しています。

ページング・アクティビティーが多い場合は、ページング・スペースの論理ボリュームが、物理ボリュームのセンターに割り当てる良い候補です。それとまったく逆に、ダンプとブートの論理ボリュームはあまり頻繁に使用されないので、物理ボリュームの先頭か末尾に割り当てます。

つまり、一般的な規則としては、絶対的にまたは重要なアプリケーションの実行で、入出力の多いものほど、論理ボリュームの物理パーティションを物理ボリュームのセンター近くに割り当てる必要があるということです。

物理ボリュームの範囲

Inter-Physical Volume Allocation Policy (物理ボリューム間割り振りポリシー) では、論理ボリュームの物理パーティションを割り当てる物理デバイスの選択に、どのストラテジーを使用するべきかを指定します。選択項目は、最小オプションと最大オプションです。

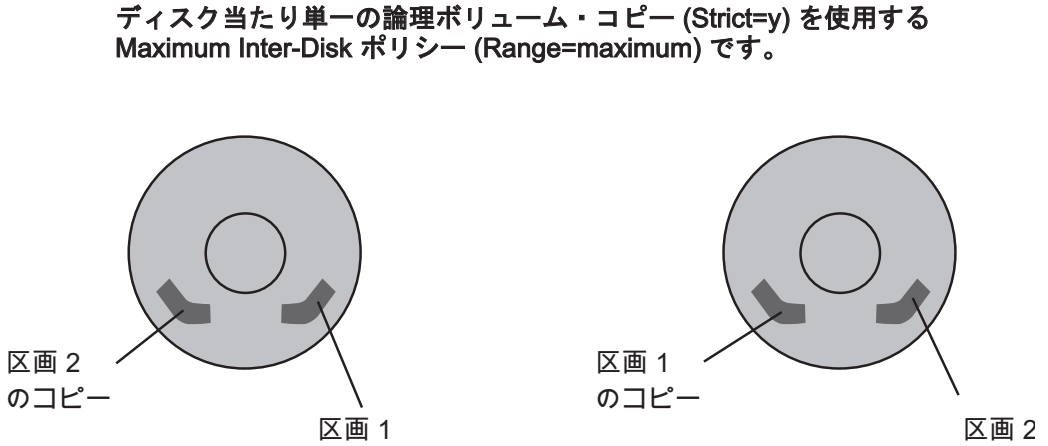


図 *18.* 物理ボリューム間割り振りポリシー： この図は、2 つの物理ボリュームを示しています。 一方には、パーティション 1 と、パーティション 2 のコピーが入っています。他方には、パーティション 2 と、パーティション 1 のコピーが入っています。割り当ての公式は、ディスク当たり単一の論理ボリューム・コピー (Strict=y) を使用する Maximum Inter-Disk ポリシー (Range=maximum) です。

最小オプションは、必須の物理パーティションの割り当てに使用される物理ボリュームの数を示します。これは一般に、コピーせずに、論理ボリュームに最大の信頼性と可用性を提供するために使用するポリシーです。 最小オプションを使用するときは、次のようにコピーありとコピーなしの 2 つの選択項目があります。

- コピーなし: 最小オプションは、1 つの物理ボリュームにこの論理ボリュームのすべての物理パーティションを入れることを示します。割り当てプログラムは、2 つ以上の物理ボリュームを使用する必要がある場合は、他のパラメーターとの整合性を保ったまま、可能な最小数を使用します。
- コピーあり: 最小オプションは、物理ボリュームはコピーがある限りすべて使用することを示します。割り当てプログラムは、2 つ以上の物理ボリュームを使用する必要がある場合は、可能な最小数を使用して、すべての物理パーティションを保持します。 常に、strict オプションなどの、他のパラメーターによる制約を監視しています。

これらの定義は、既存の論理ボリュームの拡張またはコピー時に適用できます。 既存の割り当ては、例えば、コピーありのケースで使用する物理ボリュームの数の判別の際のカウントに入っています。

最大オプションは、必須の物理パーティションの割り当てに使用される物理ボリュームの数を示します。最大オプションは、他の制約も考慮しながら、この論理ボリュームの物理パーティションを、できる限り多くの物理ボリュームに分散することを意図しています。 これはパフォーマンス指向のオプションで、コピーありで使用して、可用性を改善します。 コピーされていない論理ボリュームを複数の物理ボリュームに分散した場合、その論理ボリュームの物理パーティションを含む物理ボリュームが失われたりすると、それだけで論理ボリュームが不完全になってしまいます。

割り当てに使用する物理ボリュームの最大数

新規割り当て用の物理ボリュームの最大数を設定します。

この値は、1 とボリューム・グループ内の物理ボリューム数の間でなければなりません。 このオプションは 218 ページの『物理ボリュームの範囲』と関係があります。

ミラー書き込み整合性

LVM は、通常の入出力処理の間、常にミラーリングされた論理ボリュームのコピー間のデータ整合性を保証します。

論理ボリュームへの書き込みの際には、LVM がすべてのミラー・コピーの書き込み要求を生成します。ミラーリングされた書き込みの処理中 (すべてのコピーが書き込まれる前に) にシステムがクラッシュすると、問題が発生します。 論理ボリュームのミラー書き込み整合性のリカバリーが要求された場合、 LVM はこのような整合性のないミラーのリカバリーを可能にするための追加情報を保持します。 ミラー書き込み整合性のリカバリーは、大部分のミラーリングされた論理ボリュームに対して行う必要があります。ボリューム・グループが再度 varyon されるときに既存データを使用しないページ・スペースなどの論理ボリュームは、このような保護を必要としません。

ミラー書き込み整合性 (MWC) レコードは、1 つのセクターで構成されます。 これは、システムが正しくシャットダウンされないと、どのロジカル・パーティションが不整合になる可能性があるかを識別します。ボリューム・グループが再びオンラインに戻された場合、この情報が、ロジカル・パーティションに再び整合性を持たせるために使用されます。

注**:** ミラー書き込み整合性 LV では、MWC 制御セクターがディスクのエッジにあるので、ミラーリングされた論理ボリュームもエッジにある場合は、パフォーマンスが向上する可能性があります。

AIX 5 より、受動ミラー書き込み整合性と呼ばれる、ミラー書き込み整合性オプションが使用可能になりました。ミラー書き込み整合性を確実にするデフォルトのメカニズムが、アクティブ MWC です。アクティブ MWC により、クラッシュが発生した後のリブート時のリカバリーが速くなります。ただし、このメリットは書き込みのパフォーマンスを犠牲にして得られるものであり、特に、ランダム書き込みのパフォーマンスが低下します。 アクティブ MWC を使用不可にすれば、この書き込みパフォーマンスの低下はなくなりますが、クラッシュ後のリブートで **syncvg -f** コマンドを使用してボリューム・グループ全体を手作業で同期させてからでないと、ユーザーがそのボリューム・グループにアクセスすることができません。それには、ボリューム・グループの自動 vary-on を使用不可にしておく必要があります。

受動 MWC を使用可能にした場合は、アクティブ MWC にからむ書き込みパフォーマンスの低下がなくなるだけでなく、パーティションがアクセスされているときに自動的に論理ボリュームの再同期がとられます。 つまり、アドミニストレーターが手作業で論理ボリュームの同期をとる必要がなく、また、自動 varyon を使用不可にする必要もありません。受動 MWC の欠点としては、すべてのパーティションの同期が取り直されるまでは、読み取り操作が遅くなるということがあります。

論理ボリュームの作成時、または変更時に、SMIT 内でミラー書き込み整合性オプションを選択するかしないかの選択ができます。 この選択オプションが有効になるのは、論理ボリュームのミラーリング (コピー部数 > 1) が行われた場合だけです。

関連情報:

論理ボリュームのミラー書き込み整合性ポリシー

独立した PV 上の各ロジカル・パーティションのコピーの割り当て

厳格な割り振りポリシーに従うかどうかを指定します。

厳格な割り振りポリシーでは、ロジカル・パーティションの各コピーを、独立した物理ボリュームに割り当てます。 このオプションは 218 ページの『物理ボリュームの範囲』と関係があります。

再編成中の論理ボリュームの再配置

再編成時の論理ボリュームの再配置を可能にするかどうかを指定します。

ストライピングされた論理ボリュームの場合、relocate パラメーターを no (ストライピングされた論理ボリュームの場合のデフォルト) に設定する必要があります。 ご使用のシステムによっては、論理ボリュームを再配置することもできます。

ロジカル・パーティション・コピーの読み取りおよび書き込みのためのスケジューリング・ポリシー

論理ボリュームに各種のスケジューリング・ポリシーを設定することができます。

異なるスケジューリング・ポリシーのタイプは、複数のコピーを持つ論理ボリュームで、次のように使用されます。

- *parallel* ポリシーは、ディスク間で読み取りのバランスを取ります。 読み取りのたびに、システムは 1 次が使用中かどうかチェックします。 使用中でなければ、読み取りは 1 次で開始されます。 1 次が使用中の場合、システムは 2 次をチェックします。 それが使用中でなければ、読み取りは 2 次で開始されます。 2 次が使用中なら、読み取りは未解決の入出力の数が最小のコピー上で開始されます。 書き込み (複数) は並行して開始されます。
- *parallel/sequential* ポリシーは、常に 1 次コピー上で読み取りを開始します。 書き込み (複数) は並行して開始されます。
- *parallel/round robin* ポリシーは、常に 1 次コピーを最初に調べる代わりに、コピーをかわるがわる調べる点を除けば、parallel ポリシーと同様です。 この結果、一時に複数の未解決の入出力が決してない場合でも、読み取りの使用率が等しくなります。 書き込み (複数) は並行して開始されます。
- *sequential* ポリシーでは、すべての読み取りが 1 次コピーに対して出されることになります。 書き込みは、最初は 1 次ディスクに対してシリアルに発生しますが、これは、完了したのが、2 次ディスクに対して開始された 2 番目の書き込みであった場合に限ります。

物理コピーが 1 つだけのデータの場合、論理ボリュームのデバイス・ドライバーが、論理的読み取りまたは要求アドレスを物理アドレスに変換して、その要求の処理のために適切な物理デバイス・ドライバーを呼び出します。 この単一コピー・ポリシーは、書き込み要求のための正しくない Bad Block Relocation (不良ブロックの再配置) を処理し、すべての読み取りエラーを呼び出しプロセスに戻します。

parallel および parallel/round-robin などのミラーリング - スケジューリング・ポリシーでは、読み取り主体のミラーリングされた構成のパフォーマンスを、ミラーリングされていない構成と同等にすることができます。 一般に、書き込み主体のミラーリングされた構成のパフォーマンスは、ディスクを多く使用しなければ、ミラーリングされていない構成より低くなります。

書き込み検査を可能にする

論理ボリュームに対するすべての書き込みをトレース読み取り (follow-up read) によって検査するかどうかを指定します。

これを *On* に設定すると、パフォーマンスに影響します。

ストライプ・サイズ

ストライプ・サイズにアレイのディスク数を乗算した値は、ストライプ・サイズと等しくなります。 ストライプ・サイズは 2 のべき乗で 4 KB から 128 MB までです。

ストライピングされた論理ボリュームの定義時には、少なくとも物理ドライブが 2 台必要です。 パーティション内の論理ボリュームのサイズは、使用するディスク・ドライブ数の整数倍でなければなりません。
詳しい説明は、 226 ページの『論理ボリューム・ストライピングのチューニング』を参照してください。

lvmo コマンドを使用した LVM パフォーマンス・チューニング

lvmo コマンドを使用して、ボリューム・グループごとに LVM pbuf の数を管理することができます。

lvmo コマンド用のチューナブルなパラメーターは以下のとおりです。

pv_pbuf_count
物理ボリュームをボリューム・グループに追加したときに追加される pbuf の数。

max_vg_pbuf_count
ボリューム・グループに割り当てることができる pbuf の最大数。この値を有効にするには、ボリューム・グループを一度オフに変更してからオンに変更し直す必要があります。

global_pbuf_count
物理ボリュームを任意のボリューム・グループに追加したときに追加される pbuf の最小数。 この値を変更するには、**ioo** コマンドを使用します。

aio_cache_pbuf_count
ボリューム・グループの aio_cache 論理ボリュームで使用可能な pbuf の現在の総数。ボリューム・グループに割り当てることができる **aio_cache_pbuf_count** の最大数は、**max_vg_pbuf_count** パラメーターで指定します。

以下の例では、**lvmo -a** コマンドを実行すると、rootvg ボリューム・グループ内のチューナブル・パラメーターの現在値を表示します。

```
# lvmo -a

vgname = rootvg
pv_pbuf_count = 256
total_vg_pbufs = 768
max_vg_pbuf_count = 8192
pervg_blocked_io_count = 0
global_pbuf_count = 256
global_blocked_io_count = 20
aio_cache_pbuf_count = 512
```

他のボリューム・グループの現行値を表示したいときは、次のコマンドを使用します。

```
lvmo -v <vg_name> -a
```

lvmo コマンドを使用してチューナブルな値を設定するには、次の例に示されているように、等号を使用します。

注**:** 以下の例では、チューナブルな *pv_pbuf_count* が redvg ボリューム・グループで 257 に設定されています。

```
# lvmo -v redvg -o pv_pbuf_count=257

vgname = redvg
pv_pbuf_count = 257
total_vg_pbufs = 257
max_vg_pbuf_count = 263168
pervg_blocked_io_count = 0
global_pbuf_count = 256
global_blocked_io_count = 20
```

注**:** pbuf 値を増やしすぎると、パフォーマンスの低下または予期しないシステム動作が起きる場合があります。

関連情報:

lvmo コマンド

物理ボリュームに関する考慮事項

物理ボリュームに関する考慮事項を説明します。

ディスク・ドライブのパフォーマンスに関する主な問題は、アプリケーションに関連しています。つまり、小さいアクセスがたくさん行われるか (ランダム)、それとも大規模なアクセスが少し行われるか (順次) ということです。 ランダム・アクセスの場合、一般に小容量のドライブを多数使用した方がパフォーマンスが良くなります。 順次アクセスの場合はこれとは逆になります (高速のドライブを使用するか、または多数のドライブをストライピングで使用します)。

ボリューム・グループに関する推奨事項

システム管理を容易にし、パフォーマンスを良くするために、可能なら、オペレーティング・システムが最初にインストールされる物理ボリュームだけでデフォルト・ボリューム・グループ (rootvg) を構成します。

rootvg にオペレーティング・システムだけを入れておくのは、ユーザー・データを危険にさらさずに、オペレーティング・システムのアップデート、再インストール、およびクラッシュのリカバリーを達成できるので、良い決定と言えます。 変更内容に含まれるのはオペレーティング・システムだけなので、アップデートや再インストールをより迅速に実行できます。

他にユーザー・データを入れる物理ボリューム用に、1 つ以上のボリューム・グループを定義する必要があります。 ユーザー・データを代替ボリューム・グループに入れておけば、そのデータの他のシステムへのエクスポートが容易になります。

アクティビティーによって多数のログ・トランザクションが生成される場合、きわめてアクティブなファイルシステムを 1 つのディスクに置き、そのファイルシステムに関するログは別のディスクに置いてください。ログ論理ボリューム (JFS ログまたはデータベース・ログ) にキャッシュ・デバイス (ソリッド・ステート・ディスクまたは書き込みキャッシュ付きディスク・アレイなど) を割り当てると、非常に良いパフォーマンスを得ることができます。

関連概念:

276 ページの『ファイルシステム・ログおよびログ論理ボリュームの再編成』
ジャーナル・ファイルシステム (JFS) および拡張ジャーナル・ファイルシステム (JFS2) では、データベース・ジャーナリング手法を使用して、整合性のあるファイルシステム構造を維持しています。 これには、ファイルシステム・メタデータに対して行われたトランザクションを循環式のファイルシステム・ログに複写することも含まれます。 ファイルシステム・メタデータには、スーパーブロック、i ノード、間接データ・ポインター、およびディレクトリーが含まれます。

ミラーリング rootvg のパフォーマンスへの影響

ミラーリングでは、書き込みを行う場合は、すべての論理ボリュームのコピーに対して行う必要があります。 通常、この書き込みは、ミラーリングされない論理ボリュームへの書き込みより時間がかかります。

ミラーリングは、特にデータベース環境ではカスタマー・データの保管によく使われていますが、システム・ボリュームにはあまり使用されていません。

さらに、ミラーリングではディスク入出力が 2 回行われるので、1 回の場合より多くの命令を完了する必要があり、そのため、追加のプロセッサー・オーバーヘッドが発生する可能性もあります。 rootvg の論理ボリュームのミラーリングにあたって、問題がどこに存在するかを推測できるように、rootvg の論理ボリュームの配置を理解しておくことが重要です。

/ の中のファイルを含む rootvg に見られる論理ボリューム、および多くの実行可能プログラムが入っている頻繁に使用される /usr/bin ファイルは、読み取り主体のデータでなければなりません。システム内の物理メモリーが現在のアクティビティー・レベルを保持するのに十分でない場合、ページング・スペースは書き込み専用にする必要があります。時々ページングが発生するのはどのシステムにおいてもよくあることですが、頻繁なページングが継続する場合は、通常応答時間が長くなります。 一般にメモリーを追加すれば、この問題は解決されます。

/tmp および /var ファイルシステムでは、多くのアプリケーションでファイル書き込みアクティビティーが発生します。 コンパイラーなどのアプリケーションでは通常、一時ファイルを作成して /tmp ディレクトリーに書き込みます。 /var ディレクトリーは、メール・キューおよびプリンター・キューあてのファイルを受け取ります。 **jfslog** は、通常の操作時は書き込み専用です。その他のファイルシステムの中で、通常の操作時にアクティブになっているのは /home ディレクトリーだけです。ユーザー・ホーム・ディレクトリーは、rootvg の管理を単純化するために、高い頻度で他のファイルシステムに配置されます。

rootvg は、rootvg 内の各論理ボリュームを **mklvcopy** コマンドでミラーリングすることにより、または **mirrorvg** コマンドを使用してボリューム・グループ全体をミラーリングすることにより、ミラーリングが可能です。

デフォルトでは、**mirrorvg** コマンドは parallel スケジューリング・ポリシーを使用して、すべての論理ボリュームの書き込み検査をオフにしておきます。 これによって、ページ・スペースのミラー書き込み整合性は使用可能にはなりません。 他のすべての論理ボリュームのミラー書き込み整合性は使用可能になります。 頻繁に書き込まれる論理ボリュームは、論理ボリュームとミラー書き込み整合性キャッシュの間のシーク距離が最短になるように、ディスクの外側のエッジ近くに配置してください。

rootvg のミラーリングが、パフォーマンスに大きな影響を与えることはありません。つまり、ページング・スペースをミラーリングした場合に、スローダウンが直接ページング率に関係してきます。そのため、高いページング率をサポートする構成になっていて、ページ・スペースが rootvg にあるシステムでは、rootvg のミラーリングはインプリメントしない方がよい場合があります。

要約すると、ミラーリングされた rootvg は、高ページング率が持続するワークロードでない場合は、考慮の価値があるということです。

論理ボリュームの再編成

ボリュームが非常にフラグメント化されていて、再編成が必要であることが分かった場合、**reorgvg** コマンド (または **smitty reorgvg**) を使用して、論理ボリュームを再編成し、定められたポリシーに従わせることができます。

このコマンドは、論理ボリュームの特性にしたがって、ボリューム・グループ内の物理パーティションの配置を再編成します。 このコマンドで論理ボリューム名が指定されている場合、そのリストの最初の論理ボリュームに最高の優先順位が与えられます。 このコマンドを使用するには、ボリューム・グループをオンに変更して、空きパーティションを持たせる必要があります。 再編成を行うには、各論理ボリュームの再配置可能フラグを yes に設定する必要があります。yes に設定されていない論理ボリュームは無視されます。

論理ボリュームの使用パターンが分かっていれば、各ボリュームに設定するポリシーを正しく決定できるようになります。 以下にガイドラインを示します。

- ホットな LV は異なる PV に割り当てる。
- ホットな LV は複数の PV に分散させる。

- 最もホットな LV は、ミラー書き込み整合性検査がオンになっている LV を除いて、PV のセンターに配置する。
- 最もコールドな LV は PV のエッジに配置する (順次アクセス時を除く)。
- LV は連続させる。
- LV は必要になると思われる最大サイズに定義する。
- 頻繁に使用される論理ボリュームは互いに近くに配置する。
- 順次ファイルはエッジに配置する。

最高のパフォーマンスを得るための推奨事項

パフォーマンスが高くなるように論理ボリュームを構成すると、常に可用性が低下する可能性があります。それぞれの環境において、パフォーマンスと可用性のどちらが重要かを決定してください。

SMIT コマンドで最高のパフォーマンスが得られるように構成するには、以下のガイドラインを使用します。

- 主に読み取りを行うシステムの場合、スケジューリング・ポリシーを parallel に設定してミラーリングを行えば、読み取り入出力が一番ビジーでないコピーに送られるので、よりよいパフォーマンスに得ることができます。 書き込みの実行中にミラーリングを行うと、ミラー書き込み整合性レコードの更新ばかりでなく、複数のコピーを書き込むことになるので、パフォーマンスが低下します。 割り振りポリシーを Strict に設定して、それぞれのコピーを別の物理ボリュームに置くこともできます。
- 書き込み検査ポリシーを No に設定し、コピーの数が 1 より大きい場合は、ミラー書き込み整合性を Off に設定します。
- 一般に、最も頻繁にアクセスされる論理ボリュームは、シーク時間を最小にするためにセンターに置くべきですが、次のような例外があります。
  - ディスクは、エッジの方がトラック当たりのデータをより多く保持できます。 順次方式でアクセスされる論理ボリュームは、エッジに置くとパフォーマンスが向上します。
  - もう 1 つの例外は、ミラー書き込み整合性検査 (MWCC) がオンになっている論理ボリュームです。 MWCC 制御セクターがディスクのエッジにあるので、ミラーリングされた論理ボリュームもエッジにある場合は、パフォーマンスが向上する可能性があります。
- 頻繁にアクセスされるかまたは同時にアクセスされる論理ボリュームは、ディスク上で互いに近くに置きます。 参照の位置関係は、センターに置くことより重要です。
- 中程度に使用されている論理ボリュームはミドルに置き、めったに使用されない論理ボリュームはエッジに置きます。
- 物理ボリューム間割り振りポリシーを最大に設定することによって、読み取りと書き込みを PV 間で共用できるようにすることもできます。

可用性を最高にするための推奨事項

可用性が最高になるようにシステムを構成するには (SMIT コマンドを使用して)、以下のガイドラインに従ってください。

- 3 つの LP コピーを使用する (2 回のミラーリング)。
- 書き込み検査を Yes に設定する。
- inter ポリシーを Minimum に設定する (ミラーリング・コピー数 = PV の #)。
- scheduling ポリシーを Sequential に設定する。
- allocation ポリシーを Strict に設定 (同一 PV 上にミラーリングしない)。
- ボリューム・グループ内に少なくとも 3 つの物理ボリュームを組み込む。

- 別のバス、アダプター、および電源装置に接続された物理ボリュームにコピーをミラーする。

少なくとも 3 つの物理ボリュームがあるので、1 つの物理ボリュームが選択不可になった場合にも、定数を維持することができます。 別のバス、アダプター、および電源を使用することによって、障害のない装置に接続されているコピーを使用することができます。

論理ボリューム・ストライピングのチューニング

ストライピング とは、論理ボリュームのデータを複数のディスク・ドライブに分散し、ディスク・ドライブの入出力容量を並行使用して、論理ボリューム上のデータにアクセスするという手法です。 ストライピングの主な目的は、大規模順次ファイルの読み取りや書き込みのパフォーマンスをさらに向上させることですが、ランダム・アクセスにとっても利点があります。

次の図は簡単な例を示しています。

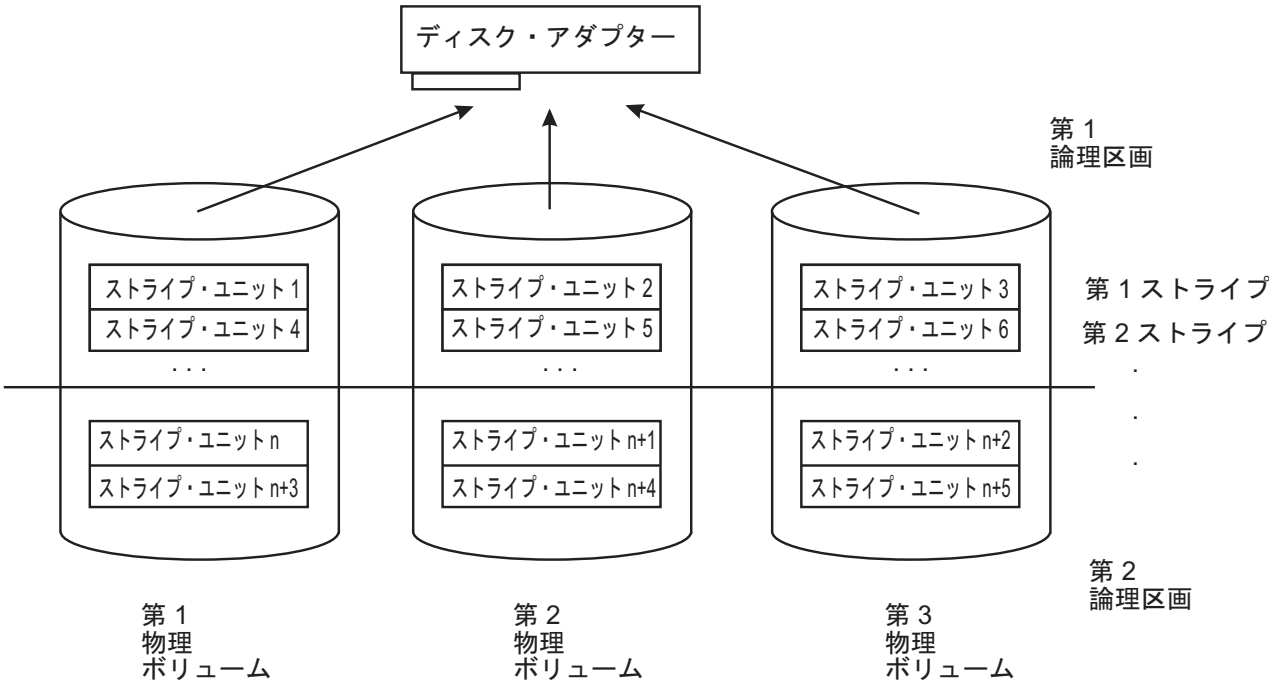


図 *19.* ストライプ論理ボリューム */dev/lvs0*： この図は、3 つの物理ボリュームまたはドライブを示しています。 それぞれの物理ドライブは、2 つの論理ボリュームに分割されています。 ドライブ 1 の最初のパーティション (論理ボリューム) にはストライプ・ユニット 1 と 4、ドライブ 2 のパーティション 1 にはストライプ・ユニット 2 と 5、ドライブ 3 のパーティション 1 にはストライプ・ユニット 3 と 6 が入っています。ドライブ 1 の 2 番目のパーティションにはストライプ・ユニット n と n+3、ドライブ 2 のパーティション 2 にはストライプ・ユニット n+1 と n+4、ドライブ 3 のパーティション 2 にはストライプ・ユニット n+2 と n+5 が入っています。

通常の論理ボリュームでは、データ・アドレスは基本的な物理パーティション内のブロックの順序に対応しています。 ストライプ論理ボリュームでは、データ・アドレスはストライプ・ユニットの順序に従っています。 完全なストライプは、ストライプ論理ボリュームの一部を含む、それぞれの物理デバイス上の 1 つのストライプ・ユニットで構成されています。 LVM は、どの物理ドライブ上のどの物理ブロックが、読み取りまたは書き込みされているブロックに対応するかを判別します。 複数のドライブが関与している場合、必要な入出力操作はすべてのドライブについて同時にスケジュールされます。

例えば、仮想の lvs0 のストライプ・ユニット・サイズが 64 KB で、6 個の 2 MB パーティションで構成されており、ジャーナル・ファイルシステム (JFS) が含まれているとします。 アプリケーションが大規

模順次ファイルを読み取っていて、先読みで定常状態に達した場合、それぞれの読み取りの結果、2 つまたは 3 つの入出力がそれぞれのディスク・ドライブに対してスケジュールされ、合計 8 ページが読み取られます (ファイルが論理ボリューム内の連続ブロックにあると想定して)。 読み取り操作はディスクのデバイス・ドライバーによって決められた順序で実行されます。 要求されたデータは入力のさまざまな断片が 1 つにまとめられて、アプリケーションに戻されます。

それぞれのディスク装置は、そのアクセス機構が操作の初めにあった位置によって、初期待ち時間はさまざまですが、プロセスが定常状態に達した後は、3 つのディスクがすべて最大スピード近くで読み取りを行っているはずです。

ストライプされた論理ボリュームの設計

ストライプされた論理ボリュームを定義するには、特定の情報を提供する必要があります。

ストライプ論理ボリュームの定義時には、次のように指定します。

ドライブ
: 少なくとも 2 つの物理ドライブ。使用されるドライブには、パフォーマンスが重要な順次入出力が発生しているときは、他の小さいアクティビティーも同時に発生します。 ディスク・アダプターとディスク・ドライブの一部の組み合わせでは、ストライプ論理ボリュームのワークロードを 2 つ以上のアダプターに分割することが必要になります。

ストライプ・ユニット・サイズ
: これは 4 KB から 128 KB の範囲の、2 の累乗の任意の数ですが、読み取りの大部分を出すメカニズムは順次先読みなので、これを考慮する必要があります。 この目的は、各先読み操作が各ディスク・ドライブに対する最低 1 つの入出力、できれば等しい数の入出力になるようにすることです (前の図を参照)。

サイズ
: 論理ボリュームに割り当てられた物理パーティションの数は、使用されるディスク・ドライブの数の整数倍でなければなりません。

属性
: ストライプ論理ボリュームのミラーリングを行うことができ、コピー数を 2 以上の値に設定できます。

ストライプ論理ボリューム入出力のためのチューニング

順次ディスク入出力およびランダム・ディスク入出力は、ディスク・ストライピングの恩恵を受けます。

次の手法は、最高レベルの順次入出力スループットをもたらしました。

- 可能な限り多くの物理ボリュームに論理ボリュームを分散する。
- 物理ボリュームに可能な限り多くのアダプターを使用する。
- ストライプ論理ボリュームに独立したボリューム・グループを作成する。
- 64 KB のストライプ・ユニットを設定する。
- **ioo** コマンドを使用して *minpgahead* を 2 に設定する。 266 ページの『順次読み取りのパフォーマンスのチューニング』を参照してください。
- **ioo** コマンドを使用して *maxpgahead* をディスク・ドライブ数の 16 倍に設定する。 これによって、ページ先読みがストライプ・ユニット・サイズ (64 KB) のディスク・ドライブ数倍の単位で行われ、その結果、各先読み操作ごとに各ディスク・ドライブから 1 つのストライプ・ユニットを読み取ることになります。
- 64 KB のディスク・ドライブ数倍の入出力を要求する。 これは *maxpgahead* 値と同じです。

- **ioo** コマンドを使用して、*maxpgahead* (*maxfree* = *minfree* + *maxpgahead*) の変更を反映するように、*maxfree* を変更する。 165 ページの『minfree および maxfree パラメーターの値』を参照してください。
- 64 バイトで位置合わせされた入出力バッファーを使用する。 論理ボリュームが 2 つ以上のディスク・アダプターに接続された物理ドライブを占有する場合は、使用する入出力バッファーは 64 バイト境界上に割り当てる必要があります。 こうすると、LVM が異なるディスクへの入出力を直列化するのを防ぐことができます。 次のコードでは、64 バイトで位置合わせされたバッファー・ポインターを生成します。

```
char *buffer;
buffer = malloc(MAXBLKSIZE+64);
buffer = ((int)buffer + 64) & ~0x3f;
```

ストライプ論理ボリュームがロー論理ボリューム上にあり、 1.125 MB を超える書き込みがそのストライピングされたロー論理ボリュームに対して行われている場合、 *lvm_bufcnt* パラメーターを **ioo** コマンドで増やすと、書き込みアクティビティーのスループットが向上する場合があります。 273 ページの『ファイルシステム・バッファーのチューニング』を参照してください。

上記の例は、JFS ストライプ論理ボリュームの場合です。 拡張 JFS にも同じ手法が適用されますが、使用する **ioo** パラメーターは、拡張 JFS の同等パラメーターになります。

また、ストライプ論理ボリュームと非ストライプ論理ボリュームを同じ物理ボリュームに混在させるのは、よい考えではありません。 すべての物理ボリュームは、1 組のストライプ論理ボリューム内では同じサイズにする必要があります。

ミラーリングされたストライプ済みの論理ボリュームのパフォーマンスへの影響

AIX では、ストライピングとミラーリングを同一論理ボリューム上に共存させることできます。

このフィーチャーでは、ハイパフォーマンスの冗長ストレージに役立つメカニズムが用意されています。 測定結果によると、ストライピングとミラーリングを使用した場合の読み取りおよびファイルシステムの書き込みのパフォーマンスが、ディスク数が 2 倍とすると、ミラーリングなしの場合とほぼ等しいことを示しています。

ファイルシステムの書き込みでは、ファイルシステム内のキャッシングが効果的で、これによって、書き込みを開始するプログラムによるディスクへの書き出しの、かなりのオーバーラップが可能になります。ただし、ロー書き込みのパフォーマンスは低下します。 これは同期しているので、開始したプログラムに制御を戻す前に、両方の書き込みが完了している必要があるからです。 大規模な書き込みを実行すると、ロー書き込みスループットは向上します。 また、ミラー書き込み整合性 (MWC) は、この場合のパフォーマンスに影響を与えます。

要約すると、ストライピングとミラーリングで実現される冗長ストレージによって、非常にハイパフォーマンスのアクセスが可能になります。

ロー・ディスク入出力の使用

データベースなどの一部のアプリケーションは、自分でロギング、データの追跡、キャッシングなどの機能を実行するため、ファイルシステムを必要としません。 ファイル入出力を使用する場合よりロー入出力を使用するときの方が、これらのアプリケーションのパフォーマンスは良くなります。メモリーのコピー、ロギング、および i ノード・ロックのための追加作業が排除されるためです。

アプリケーションがロー入出力を使用する場合、`/dev/rlv*` キャラクター特殊ファイルを使用する必要があります。 `/dev/lv*` ブロック特殊ファイルは、大きな入出力を複数の 4 K 入出力に分割するため使用しな

いでください。 /dev/rhdisk* および /dev/hdisk* ロー・ディスク・インターフェースは、パフォーマンスを低下させ、場合によってはデータ整合性の問題を発生するため、使用しないでください。

sync および fsync コール

ファイルが O_SYNC または O_DSYNC を使用してオープンされた場合は、各書き込みごとに書き込みのデータをディスクにフラッシュしてから、書き込みが終了します。書き込みによって新しいディスクの割り当てが行われる (既存のページが上書きされるのではなく、ファイルが拡張される) 場合は、対応する JFS ログの書き込みも行われます。

実メモリーとディスクの内容の同期は、次の幾つかの方法で強制的に行われます。

- アプリケーション・プログラムが指定されたファイルに対して **fsync()** を呼び出す。これにより、そのファイルの変更済みデータが含まれているすべてのページが、ディスクに書き出されます。 書き込みは、**fsync()** コールがプログラムに戻ると完了します。
- アプリケーション・プログラムが **sync()** を呼び出す。 これにより、変更済みデータが含まれているメモリー内のすべてのファイル・ページが、ディスクへの書き出しのためにスケジューリングされます。書き込みは、**sync()** コールがプログラムに戻っても、必ずしも完了するとは限りません。
- ユーザーが **sync()** コールを発行する **sync** コマンドを入力する。 この場合も、書き込みによっては、ユーザーが入力を促されても (またはシェル・スクリプト内の次のコマンドが処理されても)、完了しないことがあります。
- /usr/sbin/syncd デーモンが、一定の間隔で (通常は 60 秒ごとに) **sync()** コールを発行する。 これにより、システムは、揮発性の RAM 内にのみ存在する大量のデータを累積しなくなります。

sync 操作には、CPU の使用量が少ないことのほかにも、次のような幾つかの効果があります。

- 書き込みが拡散されず、集結する。
- 直前の **sync** 操作以降、入出力アクティビティーがなかった場合でも、28 KB 以上のシステム・データが書き込まれる。
- 遅延書き込みアルゴリズム以上に、ディスクへのデータの書き出しが加速される。 この効果は、すべての書き込み後に **fsync()** コールを発行するプログラム内で主に見られます。
- **sync()** または **fsync()** が呼び出されると、変更済みデータがディスクにコミットされたことを示すために、ログ・レコードが JFS ロギング用装置に書き込まれる。

SCSI アダプターとディスク装置キューの制限の設定

オペレーティング・システムには、SCSI アダプターから指定された SCSI バスまたはディスク・ドライブへ送られる、未解決の入出力要求の数を強制的に制限する機能があります。 この制限は、デバイス・ドライバー内のシーク最適化アルゴリズムを効果的に機能させる一方で、ハードウェアが複数の要求を処理する機能を活用することを目的としています。

IBM 以外のデバイスの場合は、最悪のケースに備えて選択されているデフォルトのキュー制限値を変更することが望ましい場合もあります。 以下のセクションでは、デフォルトを変更すべき状態と、新しい推奨値について説明します。

IBM 以外のディスク・ドライブ

IBM ディスク・ドライブの場合、ある特定の時点で未解決になり得る要求のデフォルト数は 3 です。この値は、パフォーマンスをあらゆる面から考慮して決定されたものであり、これを変更するための直接的なインターフェースはありません。 IBM 以外のディスク・ドライブの場合は、デフォルトのハードウェア・キ

ュー項目数が 1 です。特定の IBM 以外のディスク・ドライブに、複数の要求をバッファーに入れる機能が備わっている場合は、そのデバイスのシステム記述を適宜変更してください。

例として、**lsattr** コマンドを使用して表示される IBM 以外のディスク・ドライブのデフォルト特性を示します。

```
# lsattr -D -c disk -s scsi -t osdisk
pvid          none Physical volume identifier      False
clr_q         no   Device CLEARS its Queue on error
q_err         yes  Use QERR bit
q_type        none Queuing TYPE
queue_depth   1    Queue DEPTH
reassign_to   120  REASSIGN time out value
rw_timeout    30   READ/WRITE time out value
start_timeout 60   START unit time out value
```

これらのパラメーターを変更するには、SMIT (高速パスは **smitty chgdsk**) または **chdev** コマンドを使用します。 例えば、システムに IBM 以外の SCSI ディスク・ドライブ hdisk5 が内蔵されている場合は、次のコマンドによって、そのデバイスのキューイングが使用可能になり、そのキュー項目数が 3 に設定されます。

```
# chdev -l hdisk5 -a q_type=simple -a queue_depth=3
```

IBM 以外のディスク・アレイ

IBM 以外のディスク・アレイは、IBM 以外のディスク・ドライブと同様に、クラス・ディスク、サブクラス SCSI、タイプ osdisk (other SCSI disk drive (他の SCSI ディスク・ドライブ)) です。

ディスク・アレイは、オペレーティング・システムでは、大容量ディスク・ドライブとしてではなく、単一のディスク・ドライブとして扱われます。 ディスク・アレイには実際には多数の物理ディスク・ドライブが含まれ、各ディスク・ドライブが複数の要求を処理できるので、ディスク・アレイ・デバイスのキュー項目数をすべての物理デバイスを効果的に使用できる値に設定する必要があります。 例えば、hdisk7 が 8 ディスクを備えた IBM 以外のディスク・アレイである場合は、次のように変更します。

```
# chdev -l hdisk7 -a q_type=simple -a queue_depth=24
```

ディスク・アレイが SCSI-2 Fast/Wide SCSI アダプター・バスを介して接続されている場合は、そのバスの未解決の要求制限の変更が必要な場合もあります。

構成の拡張

残念なことに、パフォーマンスをあらゆる方法でチューニングしても、結果的に得られるのはわずかです。したがって問題となるのは、「必要なハードウェア、その価格、その最適な利用法」です。 この問題は、特にディスク制約のワークロードについて注意が必要です。これは変数が多数あるためです。

ディスク制約のワークロードのパフォーマンスを改善できる変更点としては、次のようなものがあります。

- ディスク・ドライブを追加し、既存のデータを分散して配置する。 これにより、ほかのアクセス機構間の入出力ロードが分散されます。
- 高速のディスク・ドライブを入手して、既存のディスク・ドライブの補完または取り替えを行い、使用率の高いデータに備える。
- 1 つ以上のディスク・アダプターを追加して、現在および新規のディスク・ドライブを接続する。
- システムに RAM を追加し、VMM の *minperm* および *maxperm* パラメーターの値を大きくして、使用率の高いデータのメモリー内キャッシングを改善する。

各種構成とワークロードに対応したガイダンスについては、測定機能付きシミュレーター (BEST/1 など) を使用してください。

RAID の使用

新磁気ディスク制御機構 (RAID) は、ディスク・アレイと各種のデータ・ストライピングという方法論を利用して、データ使用可能性を向上させる手法を説明するときに使用する用語です。

ディスク・アレイとは、単一の大容量ドライブで提供されるよりもさらに高いデータ転送速度と入出力速度を達成するために、共に機能するディスク・ドライブのグループのことです。 アレイは、複数のディスク・ドライブのセットに加えて、データを複数のドライブに分散する方法を常に把握している専用コントローラー (アレイ・コントローラー) で構成されます。 特定のファイルのデータは、単一のドライブに書き込まれるのではなく、アレイ内の複数のドライブのセグメントに書き込まれます。

アレイはまた、アレイ内の単一のドライブ (物理ディスク) に障害が起きた場合にデータが失われないように、データ冗長性も提供します。 RAID レベルに応じて、データはミラーリングされるか、またはストライピングされます。

サブアレイは、アレイ・サブシステム内に含まれています。 この構成方法に応じて、アレイ・サブシステムには 1 つ以上のサブアレイを含めることができ、これを論理装置 (LU) と呼びます。 各 LUN には独自の特性 (例えば、RAID レベル、論理ブロック・サイズおよび論理装置サイズなど) があります。 オペレーティング・システムでは、各サブアレイが固有名を持つ単一の hdisk として扱われます。

RAID アルゴリズムは、オペレーティング・システムのファイルシステム・ソフトウェアの一部として、またはディスクのデバイス・ドライバーの一部として (RAID 0 と RAID 1 に共通)、インプリメントできます。 これらのアルゴリズムは、ハードウェア RAID アダプター上のローカルに組み込まれたプロセッサーでも実行できます。 ハードウェア RAID アダプターからは一般に、ソフトウェア RAID より高いパフォーマンスを得られます。これは、組み込みプロセッサーが、複雑なアルゴリズムを実行し、場合によってはデータ転送と操作のために特殊な回路設計を採用することによって、メイン・システム・プロセッサーの負担を軽減しているからです。

LVM でサポートされる RAID オプション

AIX LVM は 3 つの RAID オプションをサポートしています。

項目	ディスクリプター
RAID 0	ストライピング
RAID 1	ミラーリング
RAID 10 または 0+1	ミラーリングとストライピング

高速書き込みキャッシュの使用法

高速書き込みキャッシュ (FWC) とは、オプションの不揮発性キャッシュのことで、標準アダプター・キャッシュに冗長性を提供します。 FWC では、ディスクにコミットされていない書き込みの状況を把握しています。

高速書き込みキャッシュを使用すると、書き込み操作に対する応答時間を大幅に短縮できます。 ただし、キャッシュがデータをデステージできる速度より速く、書き込み要求でキャッシュがオーバーフローしないように十分な注意が必要です。 また FWC は、最大入出力速度を下げることがあります。これは、転送されるデータがキャッシュ内にあるかどうかを判別するために、アダプター・カード内で別の処理が必要であるためです。

高速書き込みキャッシュは一般に、特殊なワークロード (例えば、データベースを新しいディスクのセットにコピーする場合など) には有効です。 高速書き込みキャッシュを複数のアダプターに分散すると、さらに相乗効果が期待できます。

また FWC では、JFS ログの以下のような 3 つの属性によって、JFS ログのボトルネックを解消することができます。

1. JFS ログは書き込み集中型である。 FWC は変更されていないデータをキャッシュに入れません。
2. 書き込みは量が少ないが、頻繁に行われる。 キャッシュの容量は大きくないので、高速の小さい入出力をアダプター内でより大きな物理入出力にまとめるのに最適です。 入出力が大きい場合は、データの書き込みに必要なディスク回転が通常は少なくなるので、パフォーマンスが向上する傾向があります。
3. ログは一般に、キャッシュ・サイズと比較して非常に大きいということはないので、ログがキャッシュを頻繁に「洗い流す」ということはない。 したがって、ログは既存のキャッシュ・データへの再書き込みはできません。 書き込みキャッシュを持つほかのアレイ・コントローラーはログに効果的であることが分かっていますが、ここでは、FWC に関するパフォーマンスについてだけ説明します。

単一ディスクの処理能力がパフォーマンスを制限する要因になっているときには、ソリューションの 1 つとしていくつかの RAID 5 デバイスを論理ボリュームに分割 (ストリップ) する方法があります。ストリップ・サイズは 64 KB に RAID 5 のデータ・ディスク数を乗算した値です。アダプターが RAID 5 用に構成されている場合、ストリップ・サイズ以上の大きさの書き込みでは、キャッシュをバイパスします。これが、FWC を使用した 2+p アレイへの 128 KB 書き込みが、127 KB 書き込みより遅く、FWC なしの 2+p への 128 KB 書き込みと等しくなる理由です。これは、大量の順次入出力でキャッシュを「洗い流さない」ようにすることを目的としています。

ファイバー・チャネル・デバイスの高速入出力障害

AIX は、スイッチ環境でのリンク・イベント後の、ファイバー・チャネル (FC) デバイスに対する高速入出力障害をサポートします。

FC アダプター・ドライバーはストレージ・デバイスとスイッチとの間のリンクの消失などのリンク・イベントを検出すると、ファブリックが安定できるように、15 秒ほどの短時間待機します。その時点で、デバイスがファブリック上に存在しないことを検出すると、FC アダプター・ドライバーはそのアダプター・ドライバーでのすべての入出力の失敗を開始します。 新規入出力を行ったり失敗した入出力をその後再試行しても、デバイスがファブリックを再結合したことをアダプター・ドライバーが検出するまでは、それらの入出力はアダプターにより即時失敗となります。

入出力の高速障害は、新規 **fscsi** デバイス属性 **fc_err_recov** によって制御されます。この属性のデフォルト設定は `delayed_fail` です。これは、AIX の前のバージョンで見られた入出力障害の動作です。高速入出力障害を使用可能にするには、例に示すように、この属性を `fast_fail` に設定します。

```
chdev -l fscsi0 -a fc_err_recov=fast_fail
```

この例では、**fscsi** デバイス・インスタンスは `fscsi0` です。高速障害ロジックは、アダプター・ドライバーが、スイッチからの登録状態変更通知 (RSCN) により、リモート・ストレージ・デバイス・ポートに関連したリンク・イベントがあることをスイッチから受け取ったときに呼び出されます。

マルチパス・ソフトウェアが使用されている状態では、高速入出力障害が役に立ちます。 **fc_err_recov** 属性を `fast_fail` に設定すると、ストレージ・デバイスとスイッチの間のリンク消失により、入出力障害の回数が減少することがあります。 このことは、代替パスへの高速のフェイルオーバーをサポートします。

単一パス構成、特にページング・デバイスへの単一パスがある構成では、delayed_fail デフォルト設定をお勧めします。

高速入出力障害には、以下のものが必要です。

- スイッチ環境。 公用ループを含め、アービトレーテッド・ループ環境ではサポートされません。
- FC 6227 アダプター・ファームウェア、レベル 3.22A 1 以上。
- FC 6228 アダプター・ファームウェア、レベル 3.82A 1 以上。
- FC 6239 アダプター・ファームウェア、全レベル。
- FC アダプターのすべての後続リリースでは、高速入出力障害がサポートされます。

これらのいずれかの要件が満たされていない場合、**fscsi** デバイスはこれらの要件の 1 つが満たされておらず、高速入出力障害が使用可能になっていないことを示す INFO タイプのエラー・ログを記録します。

一部の FC デバイスは、デバイスが **Available** 状態にある間の、高速入出力障害の使用可能化および使用不可化をサポートします。デバイスが動的追跡機能をサポートするかどうかを検証するには、**lsattr** コマンドを使用します。サポートするデバイスに対する高速入出力障害は、デバイスの構成解除および再構成またはリンク・サイクルなしで変更することができます。この変更は、SAN (Storage Area Network) ファブリックが安定しているときに要求する必要があります。要求時にエラー・リカバリーが SAN 内でアクティブである場合は、その要求は失敗します。

関連情報:

lsattr コマンド

ファイバー・チャネル・デバイスの動的追跡

AIX は、ファイバー・チャネル (FC) デバイスの動的追跡をサポートします。

AIX より前のバージョンでは、ユーザーは System Area Network (SAN) の設定を変更する前に FC ストレージ・デバイスとアダプター・デバイスのインスタンスを構成解除する必要があり、その結果すべてのリモート・ストレージ・ポートの N_Port ID (SCSI ID) が変更されることがありました。

FC デバイスの動的追跡が使用可能になっている場合、FC アダプター・ドライバーはデバイスのファイバー・チャネル N_Port ID が変更されると検出します。 FC アダプター・ドライバーは、そのデバイスを宛先としたトラフィックを、デバイスがまだ接続されている間に新規アドレスに転送します。 N_Port ID が変更される原因となるイベントには、次のいずれかのシナリオが含まれます。

- スイッチとストレージ・デバイスの間のケーブルを 1 つのスイッチ・ポートから別のスイッチ・ポートへ移動する。
- スイッチ間リンク (ISL) を使用して 2 つの別々のスイッチを接続する。
- スイッチをリブートする。

FC デバイスの動的追跡は、新規 **fscsi** デバイスの新規属性 dyntrk によって制御されます。 この属性のデフォルトの設定値は no です。 FC デバイスの動的追跡を使用可能にするには、例に示すように、この属性を dyntrk=yes に設定します。

```
chdev -l fscsi0 -a dyntrk=yes
```

この例では、**fscsi** デバイス・インスタンスは fscsi0 です。リモート・ストレージ・デバイス・ポートに関連したリンク・イベントがあるという通知をアダプター・ドライバーがスイッチから受け取ると、動的追跡ロジックが呼び出されます。

動的追跡サポートには、次の構成が必要です。

- スイッチ環境。 公用ループを含め、アービトレーテッド・ループ環境ではサポートされません。
- FC 6227 アダプター・ファームウェア、レベル 3.22A 1 以上。
- FC 6228 アダプター・ファームウェア、レベル 3.82A 1 以上。
- FC 6239 アダプター・ファームウェア、全レベル。
- FC アダプターの後続リリースはすべて、高速入出力障害をサポートしています。
- World Wide Name (ポート名) デバイスおよび World Wide Node Name (ノード名) デバイスは固定された値であり変更はできません。また、World Wide Name デバイスは固有でなければなりません。使用可能なデバイスまたはオンライン・デバイスの World Wide Name またはノード名を変更すると、入出力障害が起こることがあります。さらに、それぞれの FC ストレージ・デバイス・インスタンスは **world_wide_name** 属性および **node_name** 属性を持っている必要があります。 **sn_location** 属性 (次の項目を参照) を含む更新済みファイルセットも、この両方の属性を含むように更新する必要があります。
- ストレージ・デバイスは、各 LUN の固有のシリアル番号を抽出するための信頼できる方法を提供する必要があります。 AIX FC デバイス・ドライバーは、シリアル番号の位置を自動検出しません。特定デバイスの動的追跡をサポートするために、ストレージ・ベンダーはシリアル番号抽出の方法を提供する必要があります。この情報は、各ストレージ・デバイスの **sn_location** ODM 属性を使用してドライバーに伝達されます。ディスクまたはテープ・ドライバーが、**sn_location** ODM 属性が欠落していることを検出すると、`INFO` タイプのエラー・ログが生成され、動的追跡が使用不可になります。

  注**:** hdisk 上で **lsattr** コマンドを実行すると、**sn_location** 属性が表示されない場合があります。つまり、ODM 内にこの属性名があっても表示されません。
- *SAN* ファブリック 上の N_Port ID が 15 秒以内に安定した場合、FC デバイス・ドライバーは SAN ファブリック上のデバイスを追跡することができます。*SAN* ファブリック は単一のホスト・バス・アダプターから見えるファブリックです。最初の 15 秒を過ぎてもケーブルが再装着されない場合や N_Port ID が変化し続ける場合は、入出力障害が起こります。
- デバイスの追跡は、複数のホスト・バス・アダプターをまたいで行われません。 デバイスの追跡は、そのデバイスが最初に接続されたのと同じ HBA から可視のままである場合に行われます。

  例えば、デバイス A が、ホスト・バス・アダプター A に接続されているファブリック A 上のある場所から別の場所に移動された場合 (すなわち、ファブリック A での N_Port が変更された場合)、そのデバイスはユーザーの介入なしに追跡され、このデバイスの入出力を続行することができます。

  しかし、デバイス A が HBA A からは可視だが HBA B からは可視でなく、デバイス A が HBA A に接続されているファブリックから HBA B に接続されているファブリックに移動されると、デバイス A はファブリック A でもファブリック B でもアクセスできません。**cfgmgr** コマンドを実行してこのデバイスをファブリック B で使用可能にするには、ユーザー介入が必要となります。 ファブリック A の AIX デバイス・インスタンスは使用できないため、ファブリック B にデバイス・インスタンスを作成する必要があります。このデバイスは、ボリューム・グループ、マルチパス・デバイス・インスタンスなどに手動で追加する必要があります。この手順は、ファブリック A からデバイスを除去し、ファブリック B にデバイス追加することと似ています。
- AIX システム・メモリー・ダンプの進行中は、FC ダンプ・デバイスの動的追跡を行うことはできません。また、動的追跡は、システム再始動中にはサポートされず、**cfgmgr** コマンドの実行によってもサポートされません。これらの操作のいずれかが進行中であれば、SAN の変更を行うことはできません。
- デバイスの追跡を行った後、ODM 内に失効した情報が含まれている場合があります。これは、ODM 内の SCSI ID が SAN 上の実際の SCSI ID を反映しなくなるためです。 ODM は、**cfgmgr** コマンドを手動で実行するまで、または (サード・パーティーの FC SCSI ターゲット・ドライバーを含むすべてのドライバーが動的追跡をサポートしている場合に) システムをリブートするまで、このままの状

態です。 **cfgmgr** コマンドを手動で実行する場合、影響されるすべての **fscsi** デバイスを対象として実行してください。これは、オプションを指定せずに **cfgmgr** を実行するか、各 **fscsi** デバイスで個別に **cfgmgr** を実行することによって実現できます。

注**:** SCSI ID を再調整するためにランタイム時に **cfgmgr** を実行した場合、ODM 内でストレージ・デバイスの SCSI ID が更新されない可能性があります。これは、ストレージ・デバイスがオープンされている (ボリューム・グループがオンに変更された) 場合に当てはまります。オープンされていないデバイスで **cfgmgr** コマンドを実行するか、システムをリブートして SCSI ID を再調整する必要があります。 ODM 内にある失効した SCSI ID が FC ドライバーに悪影響を与えることはありません。また、FC ドライバーが正しく機能するために ODM 内の SCSI ID を再調整する必要もありません。しかし、**ioctl** 呼び出しを使用してアダプター・ドライバーと直接通信し、ODM からの SCSI ID 値を使用するアプリケーションは、失効している SCSI ID を使用するのを避けるために更新する必要があります (次の項目を参照)。

- FC 動的追跡が正しく機能するためには、**ioctl** 呼び出しを通じて、あるいは FC ドライバーのエントリー・ポイントを使用して直接、FC アダプター・ドライバーと通信するすべてのアプリケーションおよびカーネル拡張は、バージョン 1 **ioctl** および FC アダプター・ドライバーの **scsi_buf**API をサポートする必要があります。非準拠アプリケーションまたはカーネル拡張は正しく機能しなかったり、動的追跡イベント後に失敗する可能性があります。 新しいバージョン 1 **ioctl** および **scsi_buf** API に調和しないアプリケーションまたはカーネル拡張を FC アダプター・ドライバーが検出すると、INFO タイプのエラー・ログが生成され、このアプリケーションまたはカーネル拡張が通信しようとしているデバイスの動的追跡が使用可能になりません。

  AIX ファイバー・チャネル・ドライバー・スタックと通信するカーネル拡張またはアプリケーションを開発する ISV の場合、動的追跡をサポートするために必要な変更について、『必要な FCP、iSCSI、および仮想 SCSI クライアント・アダプター・デバイス・ドライバー ioctl コマンド』および『scsi_buf 構造の理解』を参照してください。
- 動的追跡が使用可能になっている場合でも、ユーザーは SAN の変更 (ケーブルの移動またはスワッピング、および ISL リンクの確立など) は、保守ウィンドウの間に行う必要があります。完全実動中に SAN の変更を行うことは、SAN の変更を行う時間のインターバルが短すぎるためお勧めできません。例えば、ケーブルが正しく再装着されていない場合、入出力障害が起こることがあります。トラフィックが少ないとき、または全くないときにこれらの操作を行うと、入出力障害の影響を最小化することができます。
- AIX 区画の動的追跡では、SAN からの構成イベントをリカバリーするという明確な目的を持つソフトウェアを使用できます。構成イベントは LPAR モビリティー操作中に発生することが予想されます。この AIX ポリシーは、モビリティー・イベントに対する準備中またはモビリティー・イベントの発生中のサービス停止を回避するために使用されます。したがって、仮想 FC クライアント・アダプター用の動的追跡は常に使用可能であり、使用不可にすることはできません。

ベースの AIX FC SCSI ディスクおよび FC SCSI テープおよび FastT デバイス・ドライバーは、動的追跡をサポートしています。 IBM ESS、EMC Symmetrix、および HDS ストレージ・デバイスは、必要な **sn_location** 属性および **node_name** 属性を持つ ODM ファイルセットをベンダーが提供している場合に、動的追跡をサポートします。現行レベルの ODM ファイルセットで動的追跡がサポートされているかどうかについては、ストレージ・ベンダーにお問い合わせください。

ベンダー固有の ODM エントリーがストレージ・デバイスに使用されていないが、ESS、Symmetrix、または HDS ストレージ・サブシステムが MPIO Other FC SCSI Disk メッセージで構成されている場合、この構成内のデバイスの動的追跡がサポートされます。この方式では **sn_location** 属性の必要性がなくなります。AIX ベースに同梱された現行の AIX Path Control Modules (PCM) はすべて、動的追跡をサポートします。

標準の AIX デバイス・ドライバーを使用する STK テープ・デバイスも、STK ファイルセットに必要な **sn_location** および **node_name** 属性が含まれている場合、動的追跡をサポートします。

注**:** テープ・デバイスに関連する SAN の変更は、アクティブな入出力がないときに行ってください。テープ・デバイスは逐次処理の性質を持っているため、1 つの入出力障害だけでアプリケーション (テープのバックアップを含む) が失敗する可能性があります。

構成時に `Other FC SCSI Disk (他の FC SCSI ディスク)` または `Other FC SCSI Tape (他の FC SCSI テープ)` メッセージが表示されたデバイスは動的追跡をサポートしません。

一部の FC デバイスは、デバイスが **Available** 状態にある間の、動的追跡の使用可能化および使用不可化をサポートします。デバイスが動的追跡をサポートするかどうかを検証するには、**lsattr** コマンドを使用します。サポートするデバイスに対する動的追跡は、デバイスの構成解除および再構成またはリンク・サイクルなしで変更することができます。この変更は、SAN (Storage Area Network) ファブリックが安定しているときに要求する必要があります。要求時にエラー・リカバリーが SAN 内でアクティブである場合は、その要求は失敗します。ディスクやテープ・デバイスなどの関連デバイスが変更に耐えることができない場合は、変更要求は失敗する可能性があります。

関連情報:

必要な FCP、iSCSI、および仮想 SCSI クライアント・アダプター (scsi_buf 構造について)

lsattr コマンド

高速入出力障害および動的追跡の相互作用

高速入出力障害とファイバー・チャネル (FC) デバイスの動的追跡は技術的には別々のフィーチャーですが、特定の状態では一方のフィーチャーを使用可能にするともう一方のフィーチャーの解釈が変更されることがあります。以下の表は、これらの設定をいろいろと置換したときに FC ドライバーが示す動作を示したものです。

dyntrk	**fc_err_recov**	**FC** ドライバーの動作
no	`delayed_fail`	デフォルト設定。 これは、AIX の前のバージョンでの従来の動作です。 デバイスの SCSI ID が変更されると、FC ドライバーはリカバリーしません。そしてリモート・ストレージ・ポートとスイッチの間でリンクの消失が発生した場合、入出力の失敗までの時間がより長くなります。動的追跡サポートが必要ないのであれば、単一パス状態ではこの設定が望ましい可能性があります。
no	`fast_fail`	ドライバーがスイッチから登録状態変更通知 (RSCN) を受信した場合、これはリモート・ストレージ・ポートとスイッチの間のリンク消失を示している可能性があります。 最初の 15 秒の遅延の後、FC ドライバーはデバイスがファブリック上にあるかを確認するための照会を行います。 ない場合、アダプターにより入出力がフラッシュバックされます。 その後の再試行または新規の入出力は、デバイスがまだファブリック上にない場合、即時失敗します。 デバイスはファブリック上に存在するが、SCSI ID が変更されていることを FC ドライバーが検出すると、FC デバイス・ドライバーはリカバリーせず、入出力は失敗して PERM エラーが戻されます。
yes	`delayed_fail`	ドライバーがスイッチから RSCN を受信した場合、これはリモート・ストレージ・ポートとスイッチの間のリンク消失を示している可能性があります。 最初の 15 秒の遅延の後、FC ドライバーはデバイスがファブリック上にあるかを確認するための照会を行います。 ない場合、アダプターにより入出力がフラッシュバックされます。 その後の再試行または新規の入出力は、デバイスがまだファブリック上にない場合、即時失敗します。ただし、ストレージ・ドライバー (ディスク・テープ、FastT ドライバー) により、入出力の再試行の間にわずかな遅延 (2-5 秒) が入れられる可能性があります。 デバイスはファブリック上に存在するが、SCSI ID が変更されていることを FC ドライバーが検出すると、FC デバイス・ドライバーはトラフィックを新規 SCSI ID に転送します。

dyntrk	fc_err_recov	FC ドライバーの動作
yes	fast_fail	ドライバーがスイッチから RSCN を受信した場合、これはリモート・ストレージ・ポートとスイッチの間のリンク消失を示している場合があります。 最初の 15 秒の遅延の後、FC ドライバーはデバイスがファブリック上にあるかを確認するための照会を行います。ない場合、アダプターにより入出力がフラッシュバックされます。その後の再試行または新規の入出力は、デバイスがまだファブリック上にない場合、即時失敗します。 ストレージ・ドライバー (ディスク、テープ、FastT) が再試行の間に遅延することはあまりありません。デバイスはファブリック上に存在するが、SCSI ID が変更されていることを FC ドライバーが検出すると、FC デバイス・ドライバーはトラフィックを新規 SCSI ID に転送します。

動的追跡が使用不可になっている場合、**fc_err_recov** 属性の delayed_fail 設定と fast_fail 設定の間には著しい相違があります。 しかし、動的追跡が使用可能になっている場合、**fc_err_recov** 属性の設定の重要度は低くなります。 この理由は、動的追跡と高速障害エラー・リカバリーのポリシーには、一部オーバーラップしている部分があるためです。 したがって、動的追跡を使用可能にすると、本質的に高速障害ロジックの一部が使用可能になります。

ファブリック上でデバイスに到達できなくなった場合の一般的なエラー回復手順は、動的追跡が使用可能になっている両方の fc_err_recov 設定で同じです。 わずかな違いは、**fc_err_recov** が delayed_fail に設定されている場合、入出力の再試行の間に遅延を注入することをストレージ・ドライバーが選択できるという点です。 これにより、入出力が永続的に失敗するまでの入出力障害の時間が、遅延の値と再試行回数に応じた追加の量だけ増加します。ただし、入出力トラフィック量が多い場合は、delayed_fail と fast_fail の間の違いはより顕著になる可能性があります。

SAN 管理者はこれらの設定を実験して、自分の環境に適した正しい設定の組み合わせを見つけることができます。

モジュラー I/O

モジュラー I/O (MIO) ライブラリーを使用すると、最良のパフォーマンスを得るために、アプリケーション・レベルでアプリケーションの入出力を分析およびチューニングすることが可能になります。

アプリケーションには、入出力パフォーマンスを最適化する機会をユーザーに与えることになる、あまり使用することのないロジックが高い頻度で組み込まれています。アプリケーション・レベルの入出力チューニングを行わないと、入出力パフォーマンスについて、エンド・ユーザーはチューニング・メカニズムの提供をオペレーティング・システムに任せることになります。通常は、指定されたシステムで複数のアプリケーションが稼働し、ハイパフォーマンス入出力結果を競い合うことになりますが、アプリケーション混合に対してはチューニング・パラメーターのセットで、せいぜい中程度のパフォーマンス結果になります。 MIO ライブラリーは、アプリケーション・レベル方式で入出力最適化のために必要なものに対応しています。

注意事項および利点

MIO を採用すると多くの利点がありますが、このライブラリーを使用する場合に注意すべきことがあります。

利点

- MIO のインプリメントは非常に簡単であり、アプリケーションの入出力分析を極めて容易にします。
- MIO によりアプリケーション・レベルでの入出力をキャッシュに入れることが可能になります。つまり、入出力システム・コールおよびそれに続くシステム割り込みを最適化することができます。

- それぞれのファイルまたはファイルのグループには pf キャッシュを構成することが可能であり、このために OS キャッシュより構成が容易になります。
- MIO は同時に実行する入出力アプリケーションで使用できます。これらのアプリケーションについては、MIO とリンクして pf キャッシュを使用したり、標準の JFS および JFS2 キャッシュをバイパスするための DIRECT I/O を使用したりするように構成することができます。 これらの MIO リンクされたアプリケーションは、MIO にリンクされていない入出力アプリケーションに対して、より多くの OS キャッシュ・スペースを解放します。
- MIO キャッシュは大規模な順次アクセス・ファイルに有効です。

注意事項

- MIO ライブラリー・キャッシュ構成を誤用すると、パフォーマンスが低下する場合があります。 これを避けるために、最初にユーザー・アプリケーションの入出力ポリシーを分析し、実際にユーザーの状態に適用するモジュールのオプション・パラメーターを見つけて、ユーザー・アプリケーションのパフォーマンス改善に役立つパラメーター値を設定します。 MIO 誤用の例:
  - OS メモリー・サイズより小さいファイルをアクセスするアプリケーションの場合、**pf** モジュールの **direct** オプションを構成すると、パフォーマンスを低下させることがあります。
  - ランダム・アクセス・ファイルの場合、キャッシュによりパフォーマンスが低下することがあります。
- MIO キャッシュは malloc サブシステムによりアプリケーションのアドレス・スペースに割り当てられますが、MIO キャッシュ・サイズの合計が使用可能な OS メモリーより大きい場合、システムはページング・スペースを使用します。 このケースでは、パフォーマンスが低下するか、またはオペレーティング・システム障害になることがあります。

MIO アーキテクチャー

モジュラー I/O (MIO) ライブラリーは、ファイル単位を基本にして実行時に起動される 5 個の I/O モジュールで構成されています。

以下のモジュールが、現在使用可能です。

- *mio* モジュール: ユーザー・プログラムへのインターフェース
- *pf* モジュール: データ・プリフェッチ・モジュール
- *trace* モジュール: 統計情報収集モジュール
- *recov* モジュール: 入出力の障害アクセスの分析と障害が発生した場合に再試行を行うモジュール
- *aix* モジュール: オペレーティング・システムへの MIO インターフェース

デフォルトのモジュールは *mio* および *aix*、その他のモジュールはオプションです。

入出力最適化と pf モジュール

pf モジュールは、ページ優先使用として簡易 LRU (Last Recently Used) メカニズムを使用するユーザー・スペース・キャッシュです。 *pf* モジュールは、これから必要になるファイル・データを予測するためにキャッシュ・ページ使用もモニターし、**aio_read** コマンドを実行してデータをキャッシュにプリロードします。

共通入出力パターンは非常に大きなファイル (10 ギガバイト単位) の順次読み取りです。 この入出力パターンを示すアプリケーションでは、オペレーティング・システムのバッファー・キャッシュの利点が最小になる傾向があります。 大規模なオペレーティング・システムのバッファー・プールは、データの再利用があったとしても非常に少ないため、効果がありません。 MIO ライブラリーは、順次アクセス・パターン

を検出し、必要になるデータが入っている多くの小さいキャッシュ・スペースを非同期でプリロードするための *pf* モジュールを呼び出して、この問題に対処するために使用できます。 *pf* キャッシュとしては、十分に先読み (プリフェッチ) を維持するだけのページを収容できる大容量のものが必要です。 *pf* モジュールは直接入出力を任意選択で使用できます。これにより、システム・バッファー・プールに別のメモリーをコピーすることを回避でき、さらに、システム・バッファーが入出力トラフィックの一回アクセスから解放され、システム・バッファーをより有効に使用できます。 AIX で JFS および JFS2 ファイルシステムを使用した早期体験では、*pf* モジュールから直接入出力を使用する大規模順次アクセス・ファイルについては、システム・スループットに非常に有効であることが、常に実証されました。

MIO インプリメンテーション

MIO のインプリメントには、**libtkio** をリンクするリダイレクト、**libmio.h** をインクルードするリダイレクト、MIO ルーチンへの明示的呼び出しの 3 つの方法があります。

上記 3 つの方法のいずれかを使用すると、インプリメンテーションは簡単ですが、中でも **libtkio** をリンクしてリダイレクトする方法をお勧めします。

tkio ライブラリーによるリダイレクト・リンク

Trap Kernel I/O (**tkio**) ライブラリーは、**libmio** パッケージに付けて納入される追加ライブラリーです。このライブラリーにより、アプリケーションに MIO 最適化を容易にインプリメントできます。

MIO をインプリメントするには、*TKIO_ALTLIB* 環境変数を使用して、基本カーネル入出力を構成し、代替ルーチン `setenv TKIO_ALTLIB "libmio.a(get_mio_ptrs.so)"` への呼び出しを可能にします。

上に述べた方法で *TKIO_ALTLIB* 環境変数を設定すると、デフォルトの共用オブジェクトを MIO 共用オブジェクト (*get_mio_ptrs.so*) で置き換えます。この結果、MIO 入出力ルーチンのすべてにポインターを付けた構造が戻されます。 ロードは最初のシステム・コールで一度だけ行われ、libtkio のエントリー・ポイントであるアプリケーションのすべての入出力システム・コールが MIO にリダイレクトされます。

MIO のインプリメントにはこの方法をお勧めします。ロードや関数呼び出しが失敗した場合、**libtkio** は正規のシステム・コールであるポインターのデフォルト構造に戻るためです。 さらに、MIO を使用するアプリケーションに問題がある場合は、*TKIO_ALTLIB* を設定しないで実行されたものから MIO を簡単に除去できます。

libmio.h によるリダイレクト

以下の MIO をインプリメントする方法では、ユーザー・アプリケーションのソース・コードに 2 行追加する必要があります。

MIO は C マクロの *USE_MIO_DEFINES* を使用してインプリメントできます。この方法では、入出力呼び出しを MIO ライブラリーにグローバルにリダイレクトするマクロ・セットを **libmio.h** ヘッダー・ファイルに定義します。 **libmio.h** ヘッダー・ファイルは **libmio** パッケージに付けて納入され、以下の **#define** ステートメントを含んでいます。

```
#define open64(a,b,c) MIO_open64(a,b,c,0)
#define close MIO_close
#define lseek64 MIO_lseek64
#define ftruncate64 MIO_ftruncate64
#define fstat64 MIO_fstat64
#define fcntl MIO_fcntl
#define ffinfo MIO_ffinfo
#define fsync MIO_fsync
#define read MIO_read
#define write MIO_write
```

```
#define aio_read64 MIO_aio_read64
#define aio_write64 MIO_aio_write64
#define aio_suspend64 MIO_aio_suspend64
#define lio_listio MIO_lio_listio
```

1. この方法を使用して MIO をインプリメントするには、ユーザーのソース・コードに 2 行追加します。

   ```
   #define USE_MIO_DEFINES
   #include "libmio.h"
   ```

2. アプリケーションを再コンパイルします。

MIO ルーチンの明示的呼び出し

MIO は MIO ルーチンを明示的に呼び出してインプリメントします。

MIO ライブラリーへの入出力呼び出しをリダイレクトするための `libmio.h` ヘッダー・ファイルと、その **#define** ステートメントの使用に代えて、**#define** ステートメントをユーザー・アプリケーションのソース・コードに直接追加できます。そして、そのソース・コードを再コンパイルします。

MIO 環境変数

MIO の構成には 4 個の環境変数があります。

MIO_STATS

診断メッセージ用のログ・ファイル、および MIO モジュールから要求された出力用のログ・ファイルを指す場合は、*MIO_STATS* を使用します。

ファイル名の解釈には 2 つの特殊なケースがあります。 ファイルが stderr または stdout のいずれかの場合、出力は該当するファイル・ストリームに対して行われます。 ファイル名が正符号 (+) で始まる場合 (例えば *+filename.txt*)、このファイル名は追加するためにオープンされ、ファイル名の先頭に符号が付けられていない場合、このファイルは上書きされます。

MIO_FILES

MIO_FILES は、**MIO_open64** が呼び出されるときに指定されるファイルに対して、どのモジュールが呼び出されるかを決定するキーを提供します。

MIO_FILES のフォーマットは、次のとおりです。

```
first_file_name_list [ module list ] second_file_name_list [ module list] ...
```

MIO_open64 が呼び出されるときに、MIO は *MIO_FILES* 環境変数が存在していることを検査します。環境変数が存在していれば MIO はこのデータを解析して、それぞれのファイルに対して起動すべきモジュールを決定します。

MIO_FILES は左から右へと解析されます。 先頭に左大括弧 ([) が付けられたすべての文字は *file_name_list* と見なされます。 *file_name_list* は、コロン (:) で区切られた *file_name_template* パターンのリストです。 *File_name_template* パターンは、MIO でオープンされているファイル名と突き合わせるために使用され、以下のワイルドカード文字を使用できます。

- アスタリスク (*) はディレクトリー名またはファイル名について、ゼロ個または 1 個以上の文字を突き合わせします。
- 疑問符 (?) はディレクトリー名またはファイル名について、1 個以上の文字を突き合わせします。
- 2 個のアスタリスク (**) は絶対パス名について、指定位置以後の残りのすべての文字を突き合わせします。

file_name_template パターンにスラッシュ (/) が含まれていない場合、**MIO_open64** サブルーチンに受け渡されたファイル名のすべてのパス・ディレクトリー情報は無視され、突き合わせはオープン済みのファイルのリーフ名のみに適用されます。

file_name_list にリストされたファイル名が *file_name_list* にあるいずれかの *file_name_template* パターンと一致すると、*file_name_list* のすぐ後に大括弧で示されたモジュール・リストが呼び出されます。*file_name_list* にリストされたファイル名が最初の *file_name_list* にある *file_name_template* パターンのいずれとも一致しない場合、構文解析プログラムは次の *file_name_list* に移り、そこにあるファイル名との突き合わせを試行します。ファイル名が複数の *file_name_template* パターンと一致する場合は、最初のパターンが使われます。オープンされるファイル名が、すべての *file_name_lists* のいずれの *file_name_template* パターンとも一致しない場合、このファイルは **aix** モジュールのデフォルトの起動によりオープンされます。一致する場合、モジュールは *MIO_FILES* 環境変数にリストされた関連モジュールから取り込まれます。モジュールは左から右への順序で呼びだされ、最左端はユーザー・プログラムに最も近いモジュール、最右端はオペレーティング・システムに最も近いモジュールになります。モジュール・リストが **mio** モジュールで始まっていない場合は、**mio** モジュールのデフォルト設定により、プレフィックスが環境変数に追加されます。**aix** モジュールが指定されていない場合、**aix** モジュールのデフォルト設定が環境変数に付加されます。

MIO_FILES の簡単な処理方法について例示します。

```
MIO_FILES= *.dat:*.scr [ trace ] *.f01:*.f02:*.f03 [ trace | pf | trace ]
```

MIO_open64 サブルーチンは `test.dat` ファイルをオープンし、この名前が ***.dat** *file_name_template* パターンと一致すると、**mio** モジュール、**trace** モジュール、および **aix** モジュールを起動します。

MIO_open64 サブルーチンは `test.f02` ファイルをオープンし、この名前が ***.f02**、2 番目の *file_name_list* の 2 番目の *file_name_template* パターンと一致すると、**mio** モジュール、**trace**モジュール、**pf** モジュール、**trace** モジュール、および **aix** モジュールを起動します。

環境変数へのデフォルトの呼び出しに対して、各モジュールのデフォルト・オプションがハードコーディングされています。ユーザーは関連した *MIO_FILES* モジュール・リストに値を指定して、デフォルト・オプションをオーバーライドできます。以下のコード例では、**trace** モジュールに対して統計情報をオンにして、`my.stats` ファイルに出力先を変更します。

```
MIO_FILES= *.dat : *.scr [ trace/stats=my.stats ]
```

モジュールのオプションはスラッシュで区切ります。 一部のオプションには関連した整数値またはストリング値が必要です。ストリング値が必須のオプションであり、そのストリングにスラッシュ (/) が含まれている場合は、ストリングを中括弧 {} で囲みます。整数値が必須のオプションには、ユーザーは k、m、g、または t (それぞれキロバイト、メガバイト、ギガバイト、テラバイトを表す) を付加することができます。整数値は 10 進数、8 進数、16 進数でも入力できます。整数値にプレフィックスとして 0x が使用されている場合、この整数は 16 進数として解釈されます。整数値にプレフィックスとして 0 が使用されている場合、この整数は 8 進数として解釈されます。上記の 2 つのテストに失敗した場合、この整数は 10 進数として解釈されます。

MIO_DEFAULTS

MIO_DEFAULTS 環境変数の目的は、*MIO_FILES* 環境変数に保管されたデータを読みやすくすることです。

ユーザーが複数の *file_name_list* とモジュール・リストの対に幾つかのモジュールを指定すると、*MIO_FILES* 環境変数は非常に長くなることがあります。 ユーザーがハードコーディングされたデフォルト値を同じ方法で繰り返しオーバーライドする場合は、*MIO_DEFAULTS* 環境変数を使用して、新しいデ

フォルト値をモジュールに指定するほうが簡単です。 この環境変数は、次の例のように、新しいデフォルト値を指定されたモジュールをコンマで区切ったリストです。

```
MIO_DEFAULTS = trace/events=prob.events , aix/debug
```

現在、いずれの **trace** モジュールもデフォルトの起動は、バイナリー・イベント・トレースが使用可能に設定され、prob.events ファイルに出力されます。そして、**aix** モジュールのいずれのデフォルトの起動も **debug** オプションは使用可能に設定されています。

MIO_DEBUG

MIO_DEBUG 環境変数の目的は、MIO のデバッグを支援することです。

MIO はキーワードに対しては *MIO_DEFAULTS* を検索し、オプションに対してはデバッグ出力を提供します。 使用可能なキーワードは、以下のとおりです。

ALL すべての *MIO_DEBUG* 環境変数キーワードをオンにします。

ENV 要求に一致する出力環境変数です。

OPEN
MIO_open64 サブルーチンに行われたオープン要求を出力します。

MODULES
MIO_open64 サブルーチンへの各呼び出しごとに起動されたモジュールを出力します。

TIMESTAMP
統計情報ファイルの各エントリーの先頭にタイム・スタンプを記録します。

DEF 各モジュールの定義テーブルを出力します。 このダンプはファイルがオープンされるときに、すべての MIO ライブラリー・モジュールに対して実行されます。

モジュール・オプションの定義

それぞれの MIO モジュールにはさまざまなオプションが使用可能であり、アプリケーション・レベルでのパフォーマンスのための分析と最適化に役立ちます。

MIO モジュール・オプションの定義

mio モジュールは MIO ユーザー・プログラムへのインターフェースであり、実行時にデフォルトで起動されます。

mode オープン用のファイル・アクセス・モードをオーバーライドします。

このモードは AIX オープン・システム・コールへのパラメーターとして渡されます。初期モードはソース・コードで AIX オープン・システム・コールに渡されますが、このモードに置き換えられます。

nomode
モードをオーバーライドしません。 これはデフォルト・オプションです。

direct オープン・フラグの **O_DIRECT** ビットを設定します。

nodirect
オープン・フラグの **O_DIRECT** ビットをクリアします。

osync オープン・フラグの **O_SYNC** ビットを設定します。

noosync
オープン・フラグの **O_SYNC** ビットをクリアします。

TRACE モジュール・オプションの定義

trace モジュールは MIO ユーザー・プログラムのための統計情報収集モジュールであり、オプションです。

stats{=output_file}
クローズ時の出力統計情報: トレース出力診断のためのファイル名

output_file が指定されていないか、または **mioout** (デフォルト値) の場合、**trace** モジュールは *MIO_STATS* 環境変数に定義された出力統計情報ファイルを検索します。

nostats
統計情報を出力しません。

events{=event_file}
バイナリー・イベント・ファイルを生成します。 デフォルト値は `trace.events` です。

noevents
バイナリー・イベント・ファイルを生成しません。 これはデフォルト・オプションです。

bytes 統計情報をバイト単位で出力します。 これはデフォルトの単位サイズです。

kbytes
統計情報をキロバイト単位で出力します。

gbytes
統計情報をギガバイト単位で出力します。

tbytes 統計情報をテラバイト単位で出力します。

inter 中間統計情報を出力します。

nointer
中間統計情報を出力しません。 これはデフォルト・オプションです。

PF モジュール・オプションの定義

pf モジュールは MIO ユーザー・プログラムのためのデータ・プリフェッチ・モジュールであり、オプションです。

pffw 書き込みモードでもページをプリフェッチします。

nopffw
書き込みモードではページをプリフェッチしません。 これはデフォルト・オプションです。

pf キャッシュのデフォルトの振る舞いでは、事前読み取りを起動した呼び出しがキャッシュへのユーザー書き込みからのものである場合、ページをキャッシュに事前読み取りしません。 これはすぐに上書きされてしまうページに読み取りすることは有益ではないからです。 しかし、後続のページへのユーザー書き込みが正しくない形式 (セクター境界で開始、および終了しない) になっていると、ダーティー・セクターとしてマークを付けるロジックはもはや有効ではなく、ダーティー・ページがディスクに同期をとって書き込まれる必要があり、その後、ページ全体がキャッシュに同期をとって読み込まれます。 このケースでは、同期ヒットをページ不在にするよりは、上書きされるページをそのまま非同期の読み取りにするほうが良策です。

release
グローバル・キャッシュ・ファイルの使用回数がゼロになると、グローバル・キャッシュ・ページをフリーにします。 これはデフォルト・オプションです。

norelease
グローバル・キャッシュ・ファイルの使用回数がゼロになっても、グローバル・キャッシュ・ページをフリーにしません。

release および **norelease** オプションは、ファイル使用回数がゼロになったときにグローバル・キャッシュで行われることを制御します。 デフォルトの振る舞いはグローバル・キャッシュをクローズして解放することです。 グローバル・キャッシュが複数回オープンしてクローズされると、ある時点でメモリーのフラグメント化が起こる可能性があります。 **norelease** オプションを使用すると、ファイル使用回数がゼロになっても、グローバル・キャッシュはオープン済みで使用可能な状態が続きます。

private
専用キャッシュを使用します。 キャッシュをオープンしているファイルのみ、キャッシュを使用できるという意味です。

global
グローバル・キャッシュを使用します。 複数のファイルが同じキャッシュ・スペースを使用できるという意味です。 これはデフォルト・オプションです。

たいていの場合、幾つかの理由でグローバル・キャッシュが最適のオプションであることを経験しています。 メモリー量を決定するものは、オープン済みのキャッシュ数とそのキャッシュの大きさです。 専用キャッシュを使用すると、個人では同時にアクティブになっている専用キャッシュの数を知ることができません。 グローバル・キャッシュは最大で 256 個オープンできます。 デフォルトではグローバル・オプションをゼロに設定して使用されます。これは 1 個のグローバル・キャッシュがオープンされるという意味です。 オープンされたそれぞれのグローバル・キャッシュは、ユーザーがファイルの特定グループに割り当てることができます。

asynchronous
子モジュールへの非同期呼び出しを使用します。 これはデフォルト・オプションです。

synchronous
子モジュールへの同期呼び出しを使用します。

noasynchronous
同期の別名です。

asynchronous、**synchronous**、および **noasynchronous** オプションは、キャッシュが非同期入出力呼び出しを使用してキャッシュ・データをファイルシステムからロードするかどうかを制御します。 必要であれば、デバッグまたは aio がシステムで使用可能でないときに使用されます。

direct 直接入出力を使用します。

nodirect
直接入出力を使用しません。これはデフォルト・オプションです。

?direct および **nodirect** オプションは、ファイル・オープン時の oflags に O_DIRECT ビットが OR 演算するかどうかを制御します。 **pf** キャッシュは直接入出力を行うことが可能です。キャッシュはすべてのキャッシュ・ページを 4K 境界に位置合わせし、入出力要求が確実に直接行われる必要がある正しく形式化された要求のみ実行しようとします。

bytes キャッシュ統計情報をバイト単位で出力します。 これはデフォルト・オプションです。

kbytes
キャッシュ統計情報をキロバイト単位で出力します。

mbytes
キャッシュ統計情報をメガバイト単位で出力します。

gbytes
キャッシュ統計情報をギガバイト単位で出力します。

tbytes キャッシュ統計情報をテラバイト単位で出力します。

cache_size
キャッシュの合計サイズ (バイト単位) です。 キロバイト、メガバイト、ギガバイト、およびテラバイトのサイズも認識されます。 デフォルト値は 64 キロバイトです。

page_size
各キャッシュ・ページのサイズ (バイト単位) です。 キロバイト、メガバイト、ギガバイト、およびテラバイトのサイズも認識されます。 デフォルト値は 4 キロバイトです。

prefetch
プリフェッチするページ数です。 デフォルト値は 1 です。

stride ストライド係数をページ数で設定します。 デフォルト値は 1 です。

stats{=output_file}
プリフェッチ使用率の統計情報 (**pf** 出力診断のファイル名) を出力します。

output_file が指定されていないか、または **mioout** の場合、このいずれかがデフォルト値ですが、**pf** モジュールは *MIO_STATS* 環境変数に定義された出力統計情報ファイルを検索します。

nostats
プリフェッチ使用率の統計情報を出力しません。

inter 中間プリフェッチ使用率の統計情報を **kill -SIGUSR1** コマンドで出力します。

nointer
中間プリフェッチ使用率の統計情報を出力しません。 これはデフォルト・オプションです。

inter オプションは、**kill -30** コマンドがアプリケーションで受け取られるときに、**pf** キャッシュに使用率統計情報の出力を指示します。

retain クローズしたファイル・データを後続の再オープンのために保存します。

notain
クローズしたファイル・データを後続の再オープンのために保存しません。 これはデフォルト・オプションです。

retain オプションは、ファイルが同じグローバル・キャッシュに再オープンされる場合、グローバル **pf** キャッシュに、ファイルのページが再利用されるキャッシュに保管されることを指示します。 基盤のファイルはクローズとオープンの間で変更しないでください。 キャッシュに入れられたファイルのページは、なお、ファイルがキャッシュでオープンされているかのように LRU 優先使用の対象になります。

listio listio メカニズムを使用します。

nolistio
listio メカニズムを使用しません。 これはデフォルト・オプションです。

キャッシュは、通常 listio を使用しません。listio は主にデバッグ用として使用されていました。

tag={tag string}
プレフィックス統計情報フローへの文字列です。

notag プレフィックス統計情報フローを使用しません。 これはデフォルト・オプションです。

タグ文字列は統計情報ファイルに印刷される出力のプレフィックスです。 統計情報ファイルが大きくなっている場合、このプレフィックスは関心のあるセクションの検索を容易にします。

scratch
ファイルはスクラッチ指定になり、クローズ時に削除されます。

noscratch
ファイルはフラッシュされ、クローズ時に保管されます。 これはデフォルト・オプションです。

ファイルをスクラッチとして取り扱うようキャッシュに指示します。これはファイルがクローズ時にフラッシュされないで、クローズ後にリンク解除されるという意味です。

passthru
キャッシュをパススルーするバイト範囲です。

ユーザーはキャッシュに入れないファイルの範囲を指定できます。 このデータはキャッシュをスルーして読み取り、および書き込みが行われます。 従来、同一ファイルに書き込み、ファイルの先頭にあるヘッダー情報を共有する並列アプリケーションには、これが必要でした。

RECOV モジュール・オプションの定義

recov モジュールは、失敗のケースの入出力アクセスを分析して再試行します。 これはオプションの MIO モジュールです。

fullwrite
すべての書き込みを完全にしようとします。 スペース不足で書き込みが失敗した場合、モジュールは再試行します。 これはデフォルト・オプションです。

partialwrite
すべての書き込みを完全にしようとはしません。 スペース不足で書き込みが失敗した場合、モジュールは再試行しません。

stats{=output_file}
recov メッセージの出力

output_file が指定されていないか、または **mioout** (デフォルト値) の場合、**recov** モジュールは *MIO_STATS* 環境変数に定義された出力統計情報ファイルを検索します。

nostats
recov メッセージに対してファイル・ストリームを出力しません。

command
書き込みエラーで実行されるコマンドです。 デフォルト値は `command={ls -l}` です。

open_command
接続拒否のオープン・エラーで実行されるコマンドです。デフォルト値は `open_command={echo connection refused}` です。

retry 再試行回数です。 デフォルト値は 1 です。

AIX モジュール・オプションの定義

aix モジュールはオペレーティング・システムへの MIO インターフェースであり、デフォルトで実行時に起動されます。

debug
オープンおよびクローズに対するデバッグ・ステートメントを印刷します。

nodebug
オープンおよびクローズに対するデバッグ・ステートメントを印刷しません。 これはデフォルト値です。

sector_size
固有のセクター・サイズを設定します。 設定しないと、セクター・サイズはファイルシステムのサイズと等しくなります。

notrunc
trunc システム・コールを実行しません。 このオプションは JFS O_DIRECT エラーの問題を回避する場合に必要です。

trunc trunc システム・コールを実行します。 これはデフォルト・オプションです。

MIO の使用例

MIO ライブラリーに関係のあるシナリオは数多くあります。

libtkio へのリンクによる MIO インプリメンテーションの例

MIO は入出力呼び出しを MIO ライブラリーにリダイレクトする libtkio へのリンクによってインプリメントできます。

以下のスクリプトでは MIO 環境変数を設定して、アプリケーションを Trap Kernel I/O (**tkio**) ライブラリーにリンクし、MIO を呼び出します。

```
#!/bin/csh
#
setenv TKIO_ALTLIB "libmio.a(get_mio_ptrs.so)"

setenv MIO_STATS example.stats
setenv MIO_FILES " *.dat [ trace/stats ] "
setenv MIO_DEFAULTS " trace/kbytes "
setenv MIO_DEBUG OPEN
#
cc -o example example.c  -ltkio

#
./example file.dat
```

libmio.h ヘッダー・ファイルをインクルードして行う MIO インプリメンテーションの例

MIO は入出力呼び出しを MIO ライブラリーにリダイレクトするために `libmio.h` ヘッダー・ファイルをアプリケーションのソース・ファイルに追加してインプリメントします。

以下の 2 行は `example.c` ファイルに追加するものです。

```
#define USE_MIO_DEFINES
#include "libmio.h"
```

以下のスクリプトでは MIO 環境変数を設定して、MIO ライブラリーを指定してアプリケーションをコンパイルおよびリンクし、MIO ライブラリーを呼び出します。

```
#!/bin/csh
#
setenv MIO_STATS example.stats
setenv MIO_FILES " *.dat [ trace/stats ] "
setenv MIO_DEFAULTS " trace/kbytes "
setenv MIO_DEBUG OPEN
#
cc -o example example.c -lmio
#
./example file.dat
```

MIO 診断出力ファイル

MIO ライブラリーの診断データは、**MIO_close** サブルーチンが呼び出されるときに統計情報ファイルに書き込まれます。

統計情報ファイルの名前は、**stats** オプションが `mioout` のデフォルト値に設定されている場合は、*MIO_STATS* 環境変数に定義されます。 統計情報ファイルには以下の情報を含めることができます。

- デバッグ情報
- **trace** モジュールの **stats** オプションが設定されている場合は、**trace** モジュールからの診断

  注**:** この診断データを抑止する場合は、**trace** モジュールの **nostats** オプションを使用し、**trace** モジュール診断データを他のものから分離する場合は、**trace** モジュールの **stats{=output_file}** オプションを使用します。
- **pf** モジュールの **stats** オプションが設定されている場合は、**pf** モジュールからの診断

  注**:** この診断データを抑止する場合は、**pf** モジュールの **nostats** オプションを使用し、**pf** モジュール診断データを他のものから分離する場合は、**pf** モジュールの **stats{=output_file}** オプションを使用します。
- **recov** モジュールの **stats** オプションが設定されている場合は、リカバリー・トレース・データ

  注**: recov** モジュール診断データを他のものから分離する場合は、**recov** モジュールの **stats{=output_file}** オプションを使用します。

trace モジュール診断ファイルの例**:**

trace モジュールの stat ファイルには、デバッグおよび診断データが収納されます。

ヘッダー・エレメント
- 日付
- ホスト名
- aio が使用可能または使用不可
- プログラム名
- MIO ライブラリー・バージョン
- 環境変数

デバッグ・エレメント
- すべての設定デバッグ・オプションのリスト
- DEF デバッグ・オプションが設定されている場合のすべてのモジュール定義テーブル
- OPEN デバッグが設定されている場合の MIO_open64 に作成されたオープン要求
- MODULES デバッグ・オプションが設定されている場合に起動されるモジュール

レイアウト指定のトレース・モジュールに固有のエレメント
- **TIMESTAMP** デバッグ・オプションが設定されている場合の時刻
- クローズまたは中間割り込みにおけるトレース
- module_list のトレース・モジュール位置
- 処理されたファイル名
- 比率: データ量を合計時間 (トレース・モジュールで費やされた累積時間) で除算した値

- 要求比率: データ量をファイルがオープンされていた時間の長さで除算した値 (この時間の長さにはファイルがオープン、クローズされる時間も含む)
- 現在の (トレース時の) ファイル・サイズとこのファイル処理時のファイルの最大サイズ
- ファイルシステム情報: ファイル・タイプ、セクター・サイズ
- ファイル・オープン・モードとフラグ
- 各関数ごと: この関数の呼び出し回数、およびこの関数の処理時間
- 読み取りまたは書き込み関数の場合: 詳細情報、例えば要求された情報 (読み取りまたは書き込みの要求サイズ)、合計 (読み取りまたは書き込みの実サイズ: AIX システム・コールで戻されたサイズ)、最小、および最大
- シークの場合: 平均シーク差分 (合計シーク差分とシーク・カウント)
- 読み取りまたは書き込みの場合: 中断情報、例えば中断と読み取りと書き込み時間を含む転送時のカウント、時間、および速度
- fcntl page_info 要求数: ページ

```
date
Trace on close or intermediate  :  previous module or calling program <-> next module : file name : (total transferred bytes/total time)=rate
    demand rate=rate/s=total transferred bytes/(close time-open time)
    current size=actual size of the file  max_size=max size of the file
mode=file open mode FileSystemType=file system type given by fststat(stat_b.f_vfstype) sector size=Minimum direct i/o transfer size
oflags=file open flags
open    open count          open time
fcntl      fcntl count         fcntl time
read      read count         read time   requested size      total size      minimum      maximum
aread    aread count       aread time requested size      total size      minimum      maximum
   suspend  count             time            rate
write      write count      write time  requested size       total size      minimum      maximum
seek      seek count        seek time      average seek delta
size
page    fcntl page_info count
```

サンプル

```
MIO statistics file : Tue May 10 14:14:08 2005
hostname=host1  : with Legacy aio available
Program=/mio/example
MIO library libmio.a 3.0.0.60  AIX 32 bit addressing built Apr 19 2005 15:08:17
MIO_INSTALL_PATH=
MIO_STATS       =example.stats
MIO_DEBUG       =OPEN
MIO_FILES       = *.dat [ trace/stats ]
MIO_DEFAULTS    = trace/kbytes

MIO_DEBUG OPEN =T

Opening file file.dat
   modules[11]=trace/stats
================================================================================

Trace close : program <-> aix : file.dat : (4800/0.04)=111538.02 kbytes/s
     demand rate=42280.91 kbytes/s=4800/(0.12-0.01))
     current size=0   max_size=1600
  mode =0640  FileSystemType=JFS sector size=4096
  oflags =0x302=RDWR  CREAT  TRUNC
  open              1     0.00
  write           100     0.02       1600       1600      16384      16384
  read            200     0.02       3200       3200      16384      16384
  seek            101     0.01 average seek delta=-48503
  fcntl             1     0.00
  trunc             1     0.01
  close             1     0.00
  size            100
================================================================================
```

pf モジュール診断ファイルの例**:**

pf モジュールの stat ファイルには、デバッグおよび診断データが収納されます。

エレメントおよびレイアウト

```
pf  close for <name of the file in the cache>
pf close for global or private cache <global cache number>
<number of page compute by cache_size/page-size> page of <page-size>   <sector_size> bytes per sector
<real number page not prefetch because of pffw option( suppress number of page prefetch because sector not valid)> /
                                                    <page not prefetch because of pffw option>  pages not preread for write
<number of unused prefetch>  unused prefetches out of  <number of started prefetch>  prefetch=<number of page to prefetch>
<number> of write behind
<number> of page syncs forced by ill formed writes
<number> of pages retained over close
<unit> transferred / Number of requests
program --> <bytes written into the cache by parent>/<number of write from parent> --> pf -->

                                                  <bytes written out of the cache from the child>/<number of partial page written>
program <-- <bytes read out of the cache by parent>/<number of read from parent> <-- pf <--
                                                  <bytes read in from child of the cache>/<number of page read from child>
```

サンプル

```
pf close for /home/user1/pthread/258/SM20182_0.SCR300
50 pages of 2097152 bytes 131072 bytes per sector
133/133 pages not preread for write
23 unused prefetches out of 242 : prefetch=2
95 write behinds
mbytes transferred / Number of requests
program --> 257/257 --> pf --> 257/131 --> aix
program <-- 269/269 <-- pf <-- 265/133 <-- aix
```

recov モジュール診断ファイルの例**:**

recov モジュールの stat ファイルには、デバッグおよび診断データが出力されます。

オープンまたは書き込みルーチンが失敗すると、**recov** モジュールは以下のエレメントを含む出力ファイルにコメントを出力します。

- 失敗のケースで実行するモジュールの **open_command** または **command** 値。 これらのオプションについて詳しくは、 246 ページの『RECOV モジュール・オプションの定義』を参照してください。
- エラー番号
- 再試行回数

```
15:30:00
 recov : command=ls -l file=file.dat errno=28 try=0
 recov : failure : new_ret=-1
```

MIO 構成例

MIO はアプリケーション・レベルで構成できます。

OS の構成

alot_buf アプリケーションは、以下のことを行います。

- 14 GB ファイルを書き込む。
- 100 KB バッファーを使用して 140,000 回順次書き込みを行う。
- 100 KB バッファーを使用してファイルを順次読み取る。
- 100 KB バッファーを使用してファイルを逆方向読み取りを行う。

```
# vmstat
 System Configuration: lcpu=2 mem=512MB
 kthr     memory             page              faults        cpu
 ----- ----------- ------------------------ ------------ -----------
  r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
```

```
  1  1 35520 67055   0   0   0   0    0   0 241   64  80  0  0 99  0

# ulimit -a
 time(seconds)        unlimited
 file(blocks)          unlimited
 data(kbytes)         131072
 stack(kbytes)        32768
 memory(kbytes)       32768
 coredump(blocks)     2097151
 nofiles(descriptors) 2000


# df -k /mio
Filesystem    1024-blocks      Free %Used    Iused %Iused Mounted on
/dev/fslv02      15728640  15715508    1%      231     1% /mio

# lslv fslv02
LOGICAL VOLUME:     fslv02                 VOLUME GROUP:   mio_vg
LV IDENTIFIER:      000b998d00004c00000000f17e5f50dd.2 PERMISSION:     read/write
VG STATE:           active/complete        LV STATE:       opened/syncd
TYPE:               jfs2                   WRITE VERIFY:   off
MAX LPs:            512                    PP SIZE:        32 megabyte(s)
COPIES:             1                      SCHED POLICY:   parallel
LPs:                480                    PPs:            480
STALE PPs:          0                      BB POLICY:      relocatable
INTER-POLICY:       minimum                RELOCATABLE:    yes
INTRA-POLICY:       middle                 UPPER BOUND:    32
MOUNT POINT:        /mio                   LABEL:          /mio
MIRROR WRITE CONSISTENCY: on/ACTIVE
EACH LP COPY ON A SEPARATE PV ?: yes
Serialize IO ?:     NO
```

アプリケーション **/mio/alot_buf** を分析するための **MIO** 構成

```
setenv MIO_DEBUG " OPEN MODULES TIMESTAMP"
setenv MIO_FILES "* [ trace/stats/kbytes ]"
setenv MIO_STATS mio_analyze.stats

time /mio/alot_buf
```

注: 出力診断ファイルは **debug** データおよび **trace** モジュール・データ用の `mio_analyze.stats` です。すべての値はキロバイト単位です。

注: **time** コマンドは MIO に指示して、コマンドの実行時間を通知します。

分析結果

- 実行時間は 28:06 です。
- MIO 分析診断出力ファイルは `mio_analyse.stats` です。

```
MIO statistics file : Thu May 26 17:32:22 2005
hostname=miohost  : with Legacy aio available
Program=/mio/alot_buf
MIO library libmio.a 3.0.0.60  AIX 64 bit addressing built Apr 19 2005 15:07:35
MIO_INSTALL_PATH=
MIO_STATS       =mio_analyze.stats
MIO_DEBUG       = MATCH OPEN MODULES TIMESTAMP
MIO_FILES       =* [ trace/stats/kbytes ]
MIO_DEFAULTS    =

MIO_DEBUG OPEN =T
MIO_DEBUG MODULES =T
MIO_DEBUG TIMESTAMP =T

17:32:22
Opening file test.dat
```

```
   modules[18]=trace/stats/kbytes
   trace/stats={mioout}/noevents/kbytes/nointer
   aix/nodebug/trunc/sector_size=0/einprogress=60
================================================================================

18:00:28
Trace close : program <-> aix : test.dat : (42000000/1513.95)=27741.92 kbytes/s
     demand rate=24912.42 kbytes/s=42000000/(1685.92-0.01))
     current size=14000000   max_size=14000000
  mode =0640  FileSystemType=JFS2 sector size=4096
  oflags =0x302=RDWR  CREAT  TRUNC
  open             1     0.01
  write       140000   238.16   14000000   14000000     102400     102400
  read        280000  1275.79   28000000   28000000     102400     102400
  seek        140003    11.45 average seek delta=-307192
  fcntl            2     0.00
  close            1     0.00
  size        140000
================================================================================
```

注:

- 102,400 バイトの 140,000 回書き込み
- 102,400 バイトの 280,000 回読み取り
- 速度は 27,741.92 KB/s

入出力パフォーマンス改善のための MIO 構成

```
setenv MIO_FILES "* [ trace/stats/kbytes | pf/cache=100m/page=2m/pref=4/stats/direct | trace/stats/kbytes ]"
setenv MIO_DEBUG "OPEN MODULES TIMESTAMP"
setenv MIO_STATS mio_pf.stats

time /mio/alot_buf
```

- ユーザー・アプリケーションの入出力分析には、`trace | pf | trace` モジュール・リストを使用する方法が適しています。 この方法では、ユーザーはアプリケーションが **pf** キャッシュから確認できるパフォーマンス、および **pf** キャッシュがオペレーティング・システムから確認できるパフォーマンスを入手できます。
- **pf** グローバル・キャッシュは 100 MB のサイズです。 各ページは 2 MB です。 プリフェッチされるページ数は 4 ページです。 **pf** キャッシュは非同期直接入出力システム・コールを行います。
- 出力診断ファイルには **debug** データ用、**trace** モジュール・データ用、および **pf** モジュール・データ用として `mio_pf.stats` があります。 すべての値はキロバイト単位です。

パフォーマンス・テストの結果

- 実行時刻は 15:41 です。
- MIO 分析診断出力ファイルは `mio_pf.stats` です。

```
MIO statistics file : Thu May 26 17:10:12 2005
hostname=uriage  : with Legacy aio available
Program=/mio/alot_buf
MIO library libmio.a 3.0.0.60  AIX 64 bit addressing built Apr 19 2005 15:07:35
MIO_INSTALL_PATH=
MIO_STATS       =mio_fs.stats
MIO_DEBUG       = MATCH OPEN MODULES TIMESTAMP
MIO_FILES       =* [ trace/stats/kbytes | pf/cache=100m/page=2m/pref=4/stats/direct | trace/stats/kbytes ]
MIO_DEFAULTS    =

MIO_DEBUG OPEN =T
MIO_DEBUG MODULES =T
MIO_DEBUG TIMESTAMP =T

17:10:12
```

```
Opening file test.dat
   modules[79]=trace/stats/kbytes|pf/cache=100m/page=2m/pref=4/stats/direct|trace/stats/kbytes
   trace/stats={mioout}/noevents/kbytes/nointer
 pf/nopffw/release/global=0/asynchronous/direct/bytes/cache_size=100m/page_size=2m/prefetch=4/st
ride=1/stats={mioout}/nointer/noretain/nolistio/notag/noscratch/passthru={0:0}
   trace/stats={mioout}/noevents/kbytes/nointer
   aix/nodebug/trunc/sector_size=0/einprogress=60
================================================================================

17:25:53
Trace close : pf <-> aix : test.dat : (41897728/619.76)=67603.08 kbytes/s
     demand rate=44527.71 kbytes/s=41897728/(940.95-0.01))
     current size=14000000   max_size=14000000
  mode =0640  FileSystemType=JFS2 sector size=4096
  oflags =0x8000302=RDWR  CREAT  TRUNC  DIRECT
  open              1     0.01
  ill form          0   mem misaligned 0
  write             1     0.21       1920       1920    1966080    1966080
  awrite         6835     0.20   13998080   13998080    2097152    2097152
   suspend       6835   219.01    63855.82 kbytes/s
  read              3     1.72       6144       6144    2097152    2097152
  aread         13619     1.02   27891584   27891584    1966080    2097152
   suspend      13619   397.59    69972.07 kbytes/s
  seek          20458     0.00 average seek delta=-2097036
  fcntl             5     0.00
  fstat             2     0.00
  close             1     0.00
  size           6836

17:25:53
pf close for test.dat
   50 pages of 2097152 bytes   4096 bytes per sector
   6840/6840 pages not preread for write
   7 unused prefetches out of 20459 : prefetch=4
         6835 write behinds
   bytes transferred / Number of requests
   program -->   14336000000/140000 --> pf --> 14336000000/6836   --> aix
   program <--   28672000000/280000 <-- pf <-- 28567273472/13622  <-- aix

17:25:53
pf close for global cache 0
   50 pages of 2097152 bytes   4096 bytes per sector
   6840/6840 pages not preread for write
   7 unused prefetches out of 20459 : prefetch=0
         6835 write behinds
   bytes transferred / Number of requests
   program -->   14336000000/140000 --> pf --> 14336000000/6836   --> aix
   program <--   28672000000/280000 <-- pf <-- 28567273472/13622  <-- aix

17:25:53
Trace close : program <-> pf : test.dat : (42000000/772.63)=54359.71 kbytes/s
     demand rate=44636.36 kbytes/s=42000000/(940.95-0.01))
     current size=14000000   max_size=14000000
  mode =0640  FileSystemType=JFS2 sector size=4096
  oflags =0x302=RDWR  CREAT  TRUNC
  open              1     0.01
  write        140000   288.88   14000000   14000000     102400     102400
  read         280000   483.75   28000000   28000000     102400     102400
  seek         140003    13.17 average seek delta=-307192
  fcntl             2     0.00
  close             1     0.00
  size         140000
================================================================================
```

注: プログラムは 102,400 バイトの 140,000 回書き込み、102,400 バイトの 280,000 回読み取りを実行しますが、pf モジュールは 2,097,152 バイトの 6,836 書き込み (この内の 6,835 は非同期書き込み) を実行し、2,097,152 バイトの 13,622 読み取り (この内の 13,619 は非同期読み取り) を実行します。 速度は 54,359.71 KB/s

ファイルシステムのパフォーマンス

AIX によりサポートされる幾つかのファイルシステム・タイプの構成は、全体のシステム・パフォーマンスに大きな影響があり、インストール後に変更しようとすると時間がかかります。

ファイルシステムの基本的な情報については、オペレーティング・システムおよびデバイスの管理を参照してください。

ファイルシステムのタイプ

AIX は、ローカル・ファイルシステムとリモート・ファイルシステムの 2 つのファイルシステムのタイプをサポートしています。

以下のファイルシステムは、ローカル・ファイルシステムに分類されます。

- ジャーナル・ファイルシステム
- 拡張ジャーナル・ファイルシステム
- CD ROM ファイルシステム
- RAM ディスク上のファイルシステム

以下のファイルシステムは、リモート・ファイルシステムに分類されます。

- ネットワーク・ファイルシステム
- 汎用並列ファイルシステム

ジャーナル・ファイルシステム (JFS)

ジャーナル・ファイルシステムでは、ファイルのメタデータをログに記録することにより、クラッシュが発生した後にファイルシステムの迅速なリカバリーが可能になっています。

ファイルシステム・ロギングを使用可能にすることにより、システムはファイルのメタデータのすべての変更を、ファイルシステムの予約エリアに記録します。実際の書き込み操作は、メタデータに対する変更のロギングが完了した後に行われます。

JFS は AIX の前のリリースで 32 ビット・カーネルで使用するために作成されたため、64 ビット・カーネル環境では最適な構成はできません。 しかし、AIX 6.1 以降のバージョンの 64 ビット・カーネル環境では、JFS は現在も使用できます。

拡張 JFS

拡張 JFS (JFS2) は、ネイティブ AIX ジャーナル・ファイルシステムの 1 つです。

拡張 JFS は 64 ビット・カーネル環境用のためのデフォルトのファイルシステムです。 32 ビット・カーネルのアドレス・スペースには制限があるため、拡張 JFS を 32 ビット・カーネル環境で使用することはお勧めできません。

データ・セットに対するサポートは、AIX オペレーティング・システムの一部として JFS2 に統合されています。データ・セットとは、データ管理の 1 つの単位です。これは、少なくとも 1 つのルート・ディレクトリーがあるディレクトリー・ツリーで構成されます。 管理には、新規データ・セットの作成、サーバーの集合全体でのデータ・セットの完全コピー (レプリカ) の作成と保守、または別のサーバーへのデータ・セットの移動が含まれます。データ・セットは、マウントされたファイルシステムの一部として存在する場合があります。つまり、マウントされたファイルシステム・インスタンスに複数のデータ・セットが含まれていることがあります。JFS2 では、データ・セットのサポートは **mkfs -o dm=on** コマンドを使用す

ることによって使用可能になります。デフォルトでは、データ・セットのサポートは使用可能になっていません。データ・セットが使用可能になった JFS2 インスタンスは、Dataset Services Manager (DSM) により管理することができます。

JFS と拡張 JFS の違い

JFS と拡張 JFS には、多くの相違点があります。

表 *4. JFS* と拡張 *JFS* の機能の違い

機能	**JFS**	拡張 **JFS**
最適化	32 ビット・カーネル	64 ビット・カーネル
最大ファイルシステム・サイズ	32 テラバイト	4 ペタバイト **注:** これはアーキテクチャー上の制限です。 AIX では現在 16 テラバイトまでしかサポートされていません。
最大ファイル・サイズ	64 ギガバイト	4 ペタバイト **注:** これはアーキテクチャー上の制限です。 AIX では現在 16 テラバイトまでしかサポートされていません。
I ノードの数	ファイルシステム作成時に固定	動的。ディスク・スペースで制限される
ラージ・ファイル・サポート	マウント・オプション	デフォルト
オンラインでのデフラグ	はい	はい
namefs	はい	はい
DMAPI	いいえ	はい
圧縮	はい	いいえ
クォータ	はい	はい
据え置き更新	はい	いいえ
直接入出力サポート	はい	はい

注**:**

- 64 ビットの JFS2 システムから 32 ビット・システムへの **mksysb** によるシステム・バックアップのクローン作成は成功しません。
- JFS ファイルシステムと異なり、JFS2 ファイルシステムでは **link()** API をバイナリー形式ディレクトリーで使用できません。 この制限により、JFS ファイルシステムでは正しく動作するある種のアプリケーションが、 JFS2 ファイルシステムでは障害が起きる場合があります。

ジャーナリング**:**

実際のデータを書き込む前に、ジャーナル・ファイルシステムがメタデータをログに記録するため、オーバーヘッドが発生して書き込みのスループットが低下します。

JFS でのパフォーマンスを改善する 1 つの方法は、**nointegrity** マウント・オプションを使用してメタデータのロギングをさせないようにすることです。 このパフォーマンスの改善は、メタデータの整合性を犠牲にして達成されることに注意してください。 したがって、このオプションを指定してマウントしたファイルシステムは、システム・クラッシュによって回復不能になる可能性があるので、このオプションは特に注意して使用してください。

JFS とちがって、拡張 JFS ではメタデータのロギングを使用不可にすることはできません。 しかし、拡張 JFS でのジャーナリングのインプリメントにより、メタデータ集中アプリケーションの処理に、より適したものになります。 したがって、パフォーマンス上の不利益は拡張 JFS では JFS の場合よりは少ないです。

ディレクトリーの編成**:**

索引ノード、つまり i ノードはすべてのファイルとディレクトリーの属性を保管するデータ構造です。 ファイルを検索する際に、プログラムはディレクトリーにあるファイル名を見て適切な i ノードを検索します。

これらの操作は頻繁に行われるため、検索に使用するメカニズムは特に重要です。

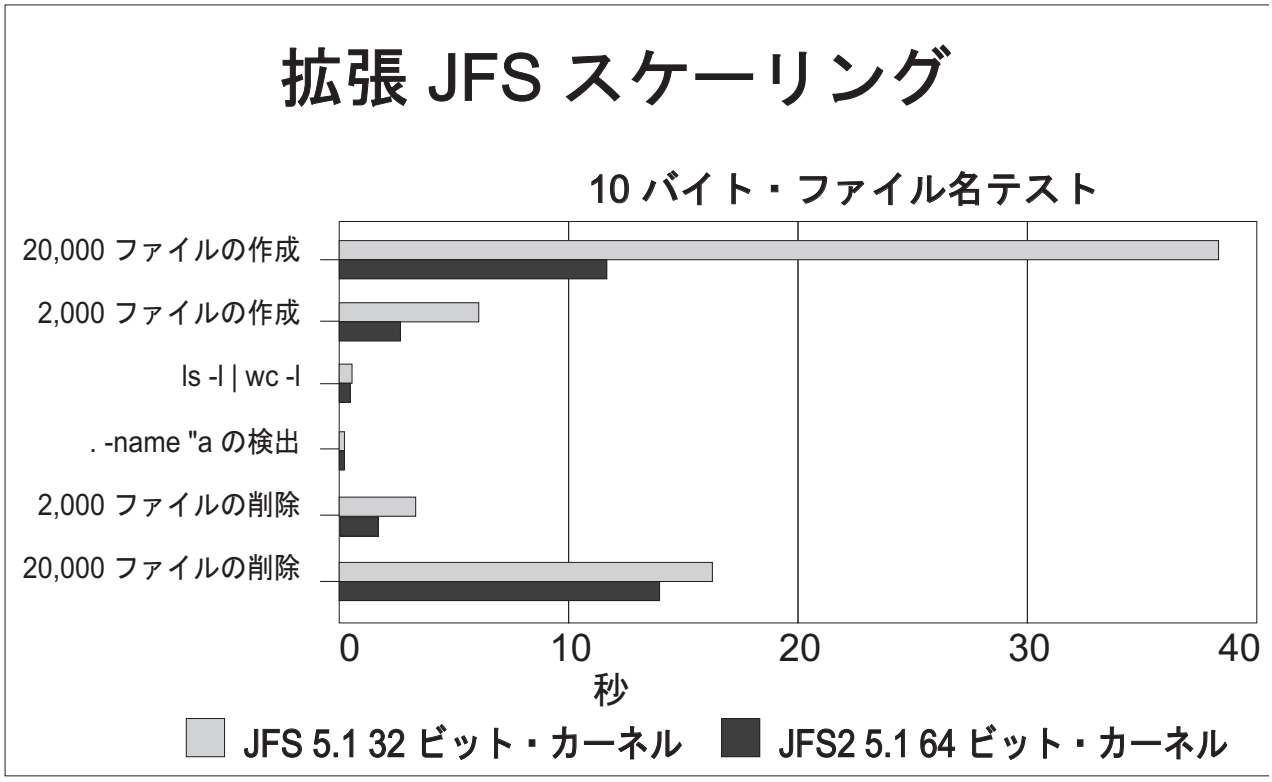
JFS はディレクトリーに直線的な編成を採用しているため、検索も直線的になります。 それに比べて、拡張 JFS はディレクトリーに二分木表現を採用しているため、ファイルへのアクセスが飛躍的に速くなっています。

スケーリング**:**

JFS と比べた拡張 JFS の主な利点は、スケーリングです。

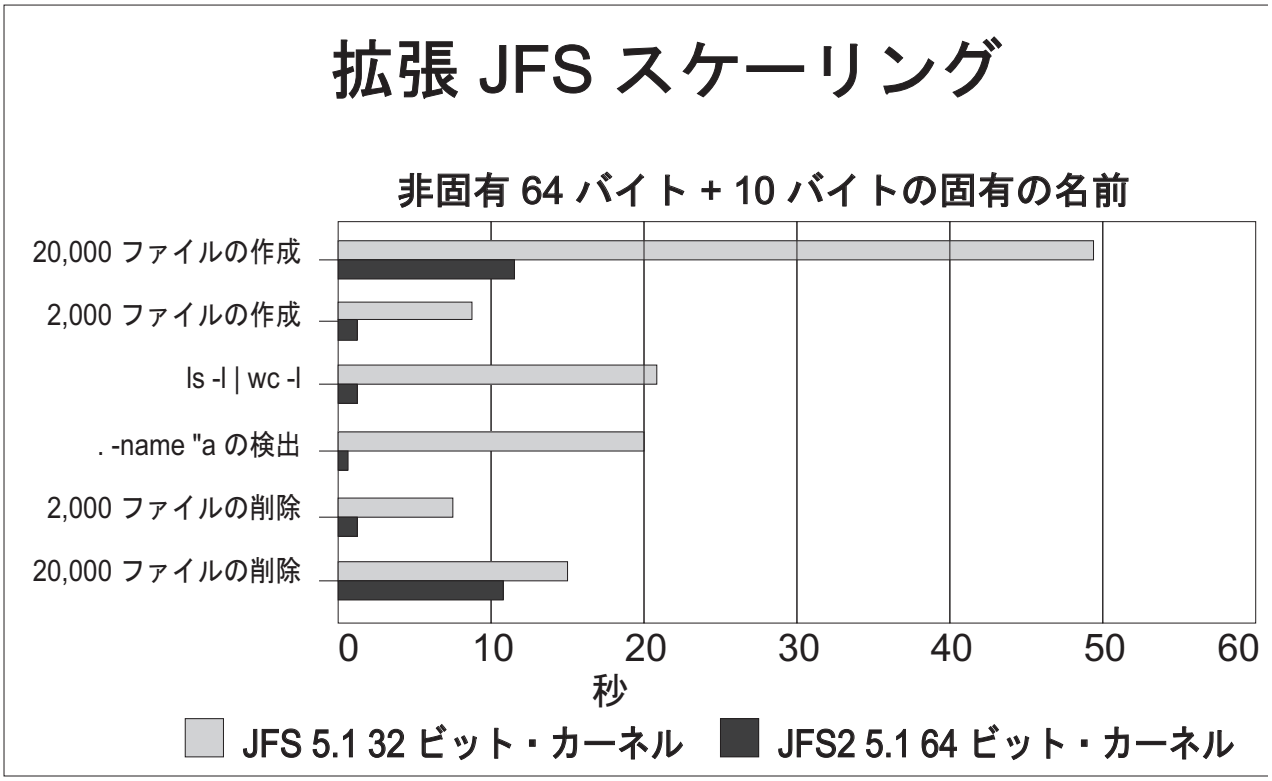
拡張 JFS は、既存の JFS よりもさらに大規模なファイルを格納する機能を備えています。 JFS でのファイルの最大サイズは 64 GB です。 AIX は、現在、拡張 JFS では最高 16 テラバイトのサイズのファイルをサポートしていますが、ファイルシステムのアーキテクチャーは、最終的に 4 ペタバイトまでのサイズのファイルを処理できるようにセットアップされます。

他のスケーリングの問題は、大量のファイルへのアクセスに関係するものです。次の図は、拡張 JFS でどのようにこの種のアクセスのパフォーマンスを改善できるかを示しています。



上記の例は、固有な 10 バイトのファイル名によるディレクトリーの作成、削除および検索で構成されています。 ファイルの作成および削除は、JFS に比べて拡張 JFS の場合の方が速いことを結果は示しています。 検索のパフォーマンスは、両方のファイルシステムでほぼ同じでした。

以下の例は、固有でないファイル名を使用した場合、JFS と比較して拡張 JFS では、一般的にどのくらい作成、削除、および検索操作が速いかを示しています。 この例では、ファイル名は最初の 64 バイトは同じで、その後に 10 バイトの固有な名前を付加しています。次の図は、このテストの結果を示しています。



関連の注釈では、長い (33 文字以上) ファイル名のキャッシングが、JFS と拡張 JFS の両方の名前のキャッシュでサポートされます。これにより、多数の長いファイル名のエントリーを含むディレクトリーに対する、**ls** や **find** コマンドなどのディレクトリー操作のパフォーマンスが改善されました。

CD ROM ファイルシステム

CD ROM ファイルシステムは、CD ROM メディア上に格納される読み取り専用のファイルシステムです。

AIX は、オペレーティング・システムおよびデバイスの管理の『ファイルシステム・タイプ』のセクションで記述されているいくつかの種類の CD-ROM ファイルシステムをサポートします。

RAM ファイルシステム

RAM ディスクとは、メモリー内にシミュレートされたディスク・ドライブです。

RAM ディスクは、物理的なドライブと比較して飛躍的に高い入出力パフォーマンスを示すように設計されていて、通常は非永続ファイルでの入出力のボトルネックを解消するために使用されます。 RAM ファイルシステムの最大サイズは、使用可能なシステム・メモリー量に制限されます。 通常のファイルシステムの使用法と同様のファイルシステムを RAM ディスク・デバイス上で作成し、使用可能にすることができます。 システムがクラッシュしたりリブートされた場合、すべてのデータが失われるため、RAM ディスクは永続的なデータには使用しないようにしてください。

ネットワーク・ファイルシステム

ネットワーク・ファイルシステム (NFS) は、ユーザーがリモート・コンピューター上にあるファイルやディレクトリーにアクセスし、それらのファイルやディレクトリーがローカルにあるかのように扱うことができる、分散ファイルシステムです。 例えば、ユーザーはオペレーティング・システム・コマンドを使用して、リモート・ファイルやディレクトリーを作成、除去、読み取り、書き込み、およびファイル属性の設定ができます。

NFS に関するパフォーマンス・チューニングとその他の問題については、『NFS パフォーマンス』というトピックを参照してください。

Name File System (NameFS)

NameFS は file-over-file および directory-over-directory マウント (ソフト・マウントとも呼ばれる) の機能を提供します。この機能を使用すると、ファイルシステムのサブツリーをファイル名スペース内の異なる場所にマウントすることができ、それにより 2 つの異なるパス名を使用してファイルにアクセスすることができます。

この機能は特定のディレクトリーの取り付け属性を変更する場合にも有用です。 例えば、特定のディレクトリー内のファイルに直接入出力サポートが必要だが、ファイルシステム全体が直接入出力に適していない場合は、**-o dio** フラグを使用 (オブジェクトを含むファイルシステムが **dio** をサポートしていると想定) して **namefs** によりそのディレクトリーまたはファイルを再マウントすることができます。

NameFS ファイルシステムは、純粋に論理上のエンティティーです。 これは、マウントされている間だけ存在し、論理上の理由のためにファイルをグループ化するための手段としてのみ使用されます。 NameFS を通じてアクセスされるオブジェクトに対するすべての操作は、そのオブジェクトを含む物理ファイルシステム・タイプによってインプリメントされ、そのファイルシステムの機能と意味はその間に介入する NameFS が存在しないかのように適用されます。

NameFS ファイルシステムは、パス名 `path1` を別のパス名 `path2` の上にマウントすることにより作成されます。 `path1` および `path2` によって指定されるオブジェクトは、通常ファイルまたはディレクトリーのいずれかであり、オブジェクトのタイプが一致している必要があります。 file-over-file マウントの後、`path2` を通じてアクセスされるオブジェクトは `path1` によって指定されるオブジェクトです。 directory-over-directory マウントの後、`path2/<pathname>` としてアクセスされるオブジェクトは、`path1/<pathname>` によって指定されるオブジェクトです。 NameFS のプログラミング・インターフェースは、標準ファイル・インターフェース・システム呼び出しによってカバーされます。 NameFS は、`vmount()` システム呼び出しに渡される **vmount** 構造に **gfstype** を指定することによりアクセスされます。 NameFS へのユーザー・インターフェースは **mount** コマンドです。 **mount** コマンドは、**vfs** タイプ *namefs* を **-v** フラグの有効なオプションとして認識します。

注**:** NameFS ファイルシステムは、NFS によってエクスポートできません。

汎用並列ファイルシステム

汎用並列ファイルシステム (すなわち、GPFS™) は、サーバー・クラスター内のすべてのノードに高速データ・アクセスを提供することのできる高性能の共用ディスク・ファイルシステムです。並列および順次アプリケーションは、AIX にある、標準の UNIX ファイルシステム・インターフェースを使用してファイルにアクセスします。

GPFS は、複数のディスクに入出力をストライピングすることによってハイパフォーマンスを提供し、ロギング、複製、およびサーバーとディスク両方のフェイルオーバーによって高可用性を提供します。

関連情報:

JFS と拡張 JFS の潜在的なパフォーマンス阻害要因

JFS および拡張 JFS のパフォーマンスを阻害する可能性のある幾つかの状態があります。

ファイルシステムのスループットに関するファイルシステム・ロギングの影響

書き込み操作はメタデータのロギングの完了後に行われるため、書き込みのスループットに影響があります。

ロギングに伴うパフォーマンス上のペナルティーの回避方法については、 193 ページの『論理ボリュームおよびディスク入出力のパフォーマンス』を参照してください。

圧縮およびフラグメント

ジャーナル・ファイルシステムは、ディスク・スペースを節約する手段として、フラグメント化した、圧縮ファイルシステムをサポートします。

平均的にはデータ圧縮ではディスク・スペースがほぼ 2 の階乗で節約できます。ただし、フラグメントと圧縮により、割り当てアクティビティーの増加に関連してパフォーマンスの低下が見られることがあります。 圧縮およびフラグメント化がパフォーマンスに与える影響については、 193 ページの『論理ボリュームおよびディスク入出力のパフォーマンス』を参照してください。

パフォーマンスを向上させるために、JFS および拡張 JFS ともにオンラインでのデフラグが可能になっています。デフラグが行われている間も、ファイルシステムのマウントおよびアクセスが可能です。

ファイルシステムのパフォーマンスの強化

AIX で、ファイルシステム・パフォーマンスを改善させるために使用できる幾つかのポリシーとメカニズムがあります。

順次ページ先読み

VMM は、プログラムがファイルにアクセスするパターンを監視することによって、ファイルのページに対する将来の要求を予想します。

プログラムがファイルの連続する 2 ページにアクセスすると、VMM は、そのプログラムがファイルの順次アクセスを継続するものと想定し、そのファイルの追加の順次読み取りをスケジュールします。 これらの読み取りはプログラムの処理とオーバーラップするので、VMM が、入出力を開始する前にプログラムが次のページにアクセスするのを待った場合よりも早く、そのプログラムでデータが使用可能になります。

JFS の場合、先読みされるページ数は、以下の VMM しきい値によって決まります。

minpgahead
VMM が初めて順次アクセス・パターンを検出したときに先読みされるページ数。

プログラムがファイルの順次アクセスを継続する場合、プログラムのアクセス後の先読みは *minpgahead* ページの 2 倍、その次の先読みは *minpgahead* ページの 4 倍、というように、ページ数が *maxpgahead* に達するまで続けられます。

maxpgahead
VMM がファイルで先読みする最大ページ数。

拡張 JFS の場合、先読みされるページ数は、以下の VMM しきい値によって決まります。

j2_minPageReadAhead
VMM が初めて順次アクセス・パターンを検出したときに先読みされるページ数。

プログラムがファイルの順次アクセスを継続する場合、プログラムのアクセス後の先読みは *j2_minPageReadAhead* ページの 2 倍、その次の先読みは *j2_minPageReadAhead* ページの 4 倍、というように、ページ数が *j2_maxPageReadAhead* に達するまで続けられます。

j2_maxPageReadAhead
VMM が順次ファイルで先読みする最大ページ数。

順次およびランダム遅延書き込み

遅延書き込みには、順次とランダムの 2 タイプがあります。

AIX ファイルシステム・コードは、以下の理由でそれぞれのファイルを JFS の場合には 16 KB のクラスター、拡張 JFS の場合には 128 KB のクラスターに論理的に分割します。

- 書き込みパフォーマンスの向上
- メモリー内のダーティー・ファイル・ページ数の制限
- システム・オーバーヘッドの削減
- ディスク・フラグメント化の最小化

指定したパーティションのページは、プログラムが次の 16 KB パーティションの最初のバイトを書き込むまで、ディスクには書き込まれません。 その時点で、ファイルシステム・コードは、最初のパーティションの 4 つのダーティー・ページを強制的にディスクに書き込みます。 データのページは、そのフレームが再利用されるまでメモリー内に残り、その時点で追加の入出力は必要ありません。 プログラムが、いずれかのページに、そのフレームが再利用される前に再アクセスする場合、入出力は必要ありません。

メモリー内に多数のダーティー・ファイル・ページが残っていて、再利用されない場合、sync デーモンはそれらのページをディスクに書き込みますが、その結果ディスクの使用状況が異常になる可能性があります。入出力アクティビティーを平等に分散させるために、遅延書き込みをオンにして、ディスクに書き出す前にメモリー内に保持するページ数を、システムに知らせることができます。遅延書き込みしきい値はファイルごとにあり、そのため、ページは sync デーモンが実行される前にディスクに書き出されます。

ioo コマンドを使用して、遅延書き込みパーティションのサイズおよび遅延書き込みのしきい値を変更することができます。 詳しくは、 267 ページの『順次およびランダム遅延書き込みのパフォーマンスのチューニング』を参照してください。

メモリー・マップ・ファイルと遅延書き込み

標準ファイルは自動的にセグメントにマップされ、マップ・ファイルを提供します。 これは、標準のファイル・アクセスが従来のカーネル・バッファーおよびブロック入出力ルーチンをバイパスして、追加のメモリーが使用可能になったとき、ファイルがさらにメモリーを使用できるようにすることを意味します。 ファイル・キャッシングは、宣言されたカーネル・バッファー域に限定されません。

ファイルは **shmat()** または **mmap()** サブルーチンによって明示的にマップすることができますが、これによってファイルのキャッシングに追加のメモリー・スペースが提供されることはありません。 **shmat()** または **mmap()** サブルーチンを使用してファイルを明示的にマップし、 **read()** および **write()** サブルーチンによってではなく、アドレスによってそのファイルにアクセスするアプリケーションは、あるパス長さのシステム・コールのオーバーヘッドを回避することができますが、システムの遅延書き込みフィーチャーの利点は失われます。

アプリケーションが **write()** サブルーチンを使用しない場合、変更されたページは、VMM ページ置換アルゴリズムまたは **sync** デーモンによって除去されると、メモリー内に累積し、ランダムに書き込まれる傾向があります。 これによって、ディスクに小さな書き込みが多数行われる結果となり、CPU やディスクの使用率を低下させるだけでなく、将来のファイルの読み取りを遅らせるフラグメント化の原因にもなります。

release-behind のメカニズム

release-behind とは、JFS および拡張 JFS 用のメカニズムです。このメカニズムの元では、ページが書き込みによって永続ストレージにコミットされるか、読み取りによってアプリケーションに引き渡されると直ちに解放されます。 このソリューションにより、近い将来に再アクセスする必要のないページを含む、非常に大きなファイルに対して順次入出力を行う際の、スケーリングの問題が解決されます。

release-behind を使用せずに大きなファイルを書き込むときは、フリー・リストに使用可能なページがあれば、書き込みは非常に高速に行われます。 ページ数がこの *minfree* パラメーター値まで減少すると、VMM は LRU アルゴリズムを使用してページアウトの候補となるページを見つけます。 この処理の一部として、VMM は書き込みにも使用されるロックを獲得する必要があります。 このロック競合が原因で、著しい性能低下となる場合があります。

mount コマンドを release-behind 順次読み取り (rbr) フラグ、release-behind 順次書き込み (rbw) フラグ、または、release-behind 順次読み取りおよび書き込み (rbrw) フラグのいずれかを指定して実行することで、release-behind を使用可能にすることができます。

release-behind メカニズム使用による副次作用は、release-behind 未使用時に比較すると、同じ読み取りまたは書き込みのスループット率に対して CPU 使用率が高くなることです。 これは、ページを解放する作業があるためです。通常、この作業は後で LRU デーモンで処理されます。 また、ファイル・ページへのアクセスは、ファイル・データが VMM によってキャッシュされないため、すべてディスク入出力となるので注意してください。

mount -o rbr コマンドを使用して、release-behind for NFS を使用できます。

直接入出力サポート

JFS および拡張 JFS は共にファイルに対する直接入出力アクセスをサポートしています。

ページをカーネル・メモリーに入れる通常のキャッシュ・ポリシーを使用する場合とは対照的に、直接入出力アクセス・メソッドはファイル・キャッシュをバイパスし、データを直接、ディスクからユーザー・スペース・バッファーに転送します。 直接入出力のチューニングについては、 274 ページの『直接入出力のチューニング』を参照してください。

遅延書き込み操作

JFS では、永続ストレージへのデータの据え置き更新が可能です。 書き込み操作を遅らせると、頻繁に再書き込みされるファイルの余分なディスク操作を減らすことができます。

遅延書き込みフィーチャーを使用可能にするには、据え置き更新フラグ **O_DEFER** を立ててファイルをオープンします。 このフィーチャーはデータをキャッシュに入れるため、他のプロセスからの読み取りおよび書き込み操作が速くなります。

このフラグでオープンされているファイルに書き込む場合は、更新されたすべてのデータを強制的にディスクにコミットする **fsync** コマンドをプロセスが出すまでは、データは永続ストレージにコミットされません。 また、ファイルに対してプロセスが同期書き込み操作を行っている、つまり、そのプロセスがファイ

ルを **O_SYNC** フラグを指定してオープンしている場合は、そのファイルが **O_DEFER** フラグを指定して作成されていても、書き込み操作が据え置かれることはありません。

注**:** このフィーチャーは、拡張 JFS では利用不能です。

並行入出力サポート

拡張 JFS は、ファイルに対する並行ファイル・アクセスをサポートします。

直接入出力のように、このアクセス・メソッドはファイル・キャッシュをバイパスし、データを直接、ディスクからユーザー・スペース・バッファーに転送します。 さらに、i ノードのロックをバイパスすることにより、複数のスレッドから同一ファイルへの読み込みおよび書き込みを並行して行えます。

注**:** このフィーチャーは、JFS では利用不能です。

パフォーマンスに影響するファイルシステム属性

ファイルシステムを長く使用すると、フラグメント化が進みます。リソースの動的割り当てによって、ファイル・ブロックはますます分散され、論理的に連続したファイルがフラグメント化され、また論理的に連続した論理ボリューム (LV) がフラグメント化されます。

フラグメント化された論理ボリュームからファイルがアクセスされた場合、以下のリストの項目が発生します。

- 順次アクセスが順次アクセスでなくなる。
- ランダム・アクセスが遅くなる。
- アクセス時間が長いシーク時間に占有される。

ただし、ファイルがメモリーに入ると、これらの影響は減少します。 ファイルシステムのパフォーマンスも、次のような物理的考慮事項による影響を受けます。

- ディスクのタイプとアダプターの数
- ファイル・バッファリングのメモリーの量
- ローカルとリモートのファイル・アクセスの量
- アプリケーションごとのファイル・アクセスのパターンと量

JFS では、スペースの使用効率を向上させるために、4 KB のブロックをさらに細分化して、ファイルシステムのフラグメント・サイズを変更することができます。 i ノードごとのバイト数 (つまり NBPI) は、1 つのファイルシステムに作成される i ノード数の制御に使用されます。 圧縮は、フラグメント・サイズが 4 KB 未満のフラグメントを持つファイルシステムに使用できます。 フラグメント・サイズと圧縮は、パフォーマンスに影響を与えます。これについては、以下のセクションで説明します。

- 『JFS のフラグメント・サイズ』
- 263 ページの『JFS の圧縮』

JFS のフラグメント・サイズ

JFS のフラグメント・フィーチャーによって、ファイルシステム内のスペースを、4 KB より小さいチャンクに割り当てることができます。

ファイルシステムを作成する際、ファイルシステム内のフラグメントのサイズを指定することができます。指定できるサイズは 512、1024、 2048、および 4096 バイトです。デフォルト値は 4096 バイトです。 フラグメントより小さいファイルは、ディスク・スペースを浪費しないために (これが主目標です)、1 つのフラグメントに一緒に保管されます。

4096 バイト未満のファイルは、必要最小数の連続フラグメントに保管されます。 サイズが 4096 バイトから 32 KB までのファイルは (両端を含む)、1 つ以上の (4 KB) フル・ブロックと残余を入れるために必要な数のフラグメントに保管されます。 例えば、5632 バイトのファイルは、i ノードの最初のポインターが指す、1 つの 4 KB ブロックに割り当てられます。フラグメント・サイズが 512 の場合、最初の 4 KB ブロックに 8 個のフラグメントが使用されます。 最後の 1.5 KB には、i ノードの 2 番目のポインターが指す、3 個のフラグメントを使用します。 32 KB より大きいファイルについては、割り当ては 4 KB ブロック単位で行われ、i ノード・ポインターはこれらの 4 KB ブロックを指します。

どのようなフラグメント・サイズであっても、フル・ブロックは 4096 バイトと見なされます。しかし、フラグメント・サイズが 4096 バイト未満のファイルシステムでは、フル・ブロックが必要な場合、合計で 4096 バイトになる、任意の連続したフラグメントで満たすことができます。 これは、4096 バイトの倍数境界から始まる必要はありません。

ファイルシステムでは、ファイル自体を論理ボリュームに分散して、ファイル間相互の割り当て妨害とフラグメント化を最小にとどめることによって、連続したフラグメントにファイルのスペースを割り当てようとします。

ファイルシステムのフラグメント・サイズが小さい場合の、パフォーマンスに関する主な障害は、スペースのフラグメント化です。 論理ボリュームに小さいファイルが分散されていると、大きいファイルを連続しているかまたは近接したブロックに割り当てることができなくなります。 ラージ・ファイルのアクセス時に、パフォーマンスが低下する恐れがあります。極端な場合、スペースのフラグメント化によって、個々の空きフラグメントは多数あるにもかかわらず、ファイルにスペースを割り当てることが不可能になる場合もあります。

ディスク入出力アクティビティーの別の逆効果には、入出力操作の数があります。サイズが 4 KB のファイルが 4 KB の 1 つのフラグメントに保管されている場合、ファイルの読み取りまたは書き込みのどちらでも、必要なのは 1 回のディスク入出力のみです。 フラグメント・サイズの選択が 512 バイトだった場合、このファイルには 8 個のフラグメントが割り当てられ、読み取りまたは書き込みが完了するには、さらに数回のディスク入出力操作 (ディスク・シーク、データ転送、および割り当てのアクティビティー) が必要になります。 したがって、4 KB のフラグメント・サイズを使用するファイルシステムでは、ディスク入出力操作の数は、フラグメント・サイズが小さいファイルシステムの場合よりはるかに少なくなる可能性があります。

小フラグメントのファイルシステムを作成する際には、その決定の一部として、**defragfs** コマンドによる、ファイルシステム内のスペースのデフラグのポリシーも考慮しておく必要があります。このポリシーでは、**defragfs** コマンドの実行のパフォーマンス・コストも考慮する必要があります。 266 ページの『ファイルシステムのデフラグ』を参照してください。

JFS の圧縮

ファイルシステムを圧縮する場合、ディスクに書き出す前に、すべてのデータが Lempel-Zev (LZ) 圧縮を使用して自動的に圧縮され、またディスクから読み取るときにすべてのデータが自動的に圧縮解除されます。 LZ アルゴリズムでは、指定された文字列の後続のオカレンスを、最初のオカレンスに対するポインターに置き換えます。平均して、50% のディスク・スペースの節約が認められます。

ファイルシステムのデータは、個々の論理ブロックのレベルで圧縮されます。データを大きい単位で圧縮すると (例えば、ファイルのすべての論理ブロックを一緒に)、より有効なディスク・スペースが失われることになります。 ファイルの論理ブロックを個別に圧縮することによって、ランダムなシークと更新が非常に迅速に実行できるようになります。

圧縮が指定されたファイルシステムにファイルを書き込む場合、圧縮アルゴリズムがデータの 4096 バイト (1 ページ) を一度に圧縮して、その後、圧縮されたデータが最小限必要な数の連続するフラグメントに書き込まれます。 明らかに、ファイルシステムのフラグメント・サイズが 4 KB の場合は、データを圧縮してもディスク・スペースの面での見返りはありません。したがって、圧縮には、4096 より小さいフラグメント・サイズのフラグメントを使用する必要があります。

圧縮すれば、全体としてスペースの浪費を防ぐことができるはずですが、ファイルシステムに多少の未使用スペースを残して置くということには、次のように正当な理由があります。

- 4096 バイト・ブロックのデータがどの程度圧縮されるかは事前には分からないので、ファイルシステムは、最初にフル・ブロックのスペースを確保しておく必要があります。 圧縮後は不要なフラグメントは解放されますが、最初の余裕を見た割り振りポリシーの結果、必要以上に早く「スペース不足」が示される可能性があります。
- **defragfs** コマンドの操作には、多少のフリー・スペースが必要です。

データ圧縮を使用するファイルシステムには、ディスク入出力アクティビティーの増加とフリー・スペースのフラグメント化の問題に加えて、次のようなパフォーマンスに関する考慮事項があります。

- データ圧縮/解凍のアクティビティー直接の結果として発生する、ファイルシステムの可用性の低下。 データの圧縮と解凍の時間がかなり長い場合、特にデータを即時に利用できることが必要な、忙しい商用環境では、常に圧縮されたファイルシステムを使用できるとは限らない場合があります。
- 圧縮されたファイルシステムのすべての論理ブロックには、最初に変更されたとき、4096 バイトのディスク・スペースが割り当てられ、このスペースは、その後論理ブロックがディスクに書き込まれるときに再割り当てされます。 パフォーマンス・コストは再割り当てと関連しており、これは非圧縮ファイルシステムでは発生しません。
- データ圧縮を実行するには、1 バイト当たりおよそ 50 CPU サイクルが必要で、解凍には 1 バイト当たり約 10 CPU サイクルが必要です。 したがって、データ圧縮を行うとプロセッサー・サイクルの数が増加し、プロセッサーに負荷を与えます。
- JFS compression kproc (jfsc) は固定優先順位 30 で実行されるため、圧縮が行われている間は、他のプロセスが (優先順位がより高いものを除いて) この kproc を実行している CPU を使用できないようになっています。

ファイルシステムの再編成

次のようにして、ファイルシステムのフラグメント化を低減することができます。

- バックアップ・メディアにファイルをコピーする。
- **mkfs** *fsname* コマンドを使用してファイルシステムを再作成するか、ファイルシステムの内容を削除する。
- ファイルシステムにファイルを再ロードする。

この手順によってファイルを順次にロードし、フラグメント化を低減します。 以下のセクションでは、詳しい情報を提供します。

- 『ファイルシステムの再編成』
- 266 ページの『ファイルシステムのデフラグ』

ファイルシステムの再編成

このセクションでは、ファイルシステムの再編成に関する手順を提供します。

以下の例では、システムには分離された論理ボリュームとファイルシステムの *hd11* (マウント・ポイント: /home/op) があります。ファイルシステム *hd11* を再編成する必要があると決定して、以下の手順を実行します。

1. ファイルシステムをファイル名でバックアップします。 名前でなく i ノードでファイルシステムをバックアップすると、 **restore** コマンドはファイルをそれぞれの元の場所に復元するので、問題を解決することにはなりません。次のコマンドを実行します。

   ```
   # cd /home/op
   # find . -print | backup -ivf/tmp/op.backup
   ```

   このコマンドは、バックアップ・ファイルを (異なるファイルシステムに) 作成し、そこに再編成するファイルシステム内のすべてのファイルを格納します。 システム上のディスク・スペースに制約がある場合、磁気テープを使用してファイルシステムをバックアップすることができます。

2. 次のコマンドを実行します。

   ```
   # cd /
   # unmount /home/op
   ```

   任意のプロセスが /home/op またはそのサブディレクトリーのどれかを使用している場合、それらのプロセスを終了しなければ、**unmount** コマンドを正常に完了することができません。

3. 次のようにして、/home/op 論理ボリューム上にファイルシステムを再作成します。

   ```
   # mkfs /dev/hd11
   ```

   古いファイルシステムを破棄する前に、確認のためのプロンプトが出されます。 ファイルシステムの名前は変更されません。

4. 元の状態に復元するには (/home/op が空である点は除いて)、以下を実行します。

   ```
   # mount /dev/hd11 /home/op
   # cd /home/op
   ```

5. 次のようにして、データを復元します。

   ```
   # restore -xvf/tmp/op.backup >/dev/null
   ```

   標準出力は、時間のかかる、復元された各ファイルの名前の表示を避けるために、/dev/null にリダイレクトされます。

6. 前に調べたラージ・ファイルを、次のようにして検討します (『fileplace コマンドによるファイル配置の評価』を参照)。

   ```
   # fileplace -piv big1
   ```

これで (ほとんど) 連続になったことが分かります。

```
File: big1  Size: 3554273 bytes  Vol: /dev/hd11
Blk Size: 4096  Frag Size: 4096  Nfrags: 868   Compress: no
Inode: 8290  Mode: -rwxr-xr-x  Owner: hoetzel  Group: system

INDIRECT BLOCK: 60307

  Physical Addresses (mirror copy 1)                          Logical Fragment

  ----------------------------------                          ----------------

  0060299-0060306  hdisk1     8 frags    32768 Bytes,   0.9%    0008555-0008562
  0060308-0061167  hdisk1   860 frags  3522560 Bytes,  99.1%    0008564-0009423

  868 frags over space of 869 frags:  space efficiency = 99.9%
  2 fragments out of 868 possible:  sequentiality = 99.9%
```

fileplace コマンドに追加した **-i** オプションは、ファイルの最初の 8 ブロックと間接ブロックを含む残りの間に 1 ブロックのギャップがあり、これはファイルの長さが 8 ブロックを超えるときに i ノード情報を補足するために必要です。

一部のファイルシステムまたは論理ボリュームは、データが一時的であるか (例えば、/tmp) またはファイルシステム・フォーマット (ログ) でないので、再編成するべきではありません。ルート・ファイルシステムは、普通は高揮発性ではないので、再編成はまれにしか必要ありません。これはインストール/保守モードのみで可能です。 /usr についても、このようなファイルは通常のシステム操作に必要なので、同じことが言えます。

ファイルシステムのデフラグ

ファイルシステムが 4 KB より小さいフラグメント・サイズで作成されている場合、ある時間過ぎた後に、分散して使用できなくなったフラグメントの量を照会することが必要になります。 多くの小さいフラグメントが分散していると、連続する使用可能なスペースを見つけることが困難になります。

このような小さい、分散したスペースを回復するには、**smitty dejfs** コマンドか **smitty dejfs2** コマンド、または **defragfs** コマンドのいずれかを使用します。 デフラグ・プロシージャーを使用するには、多少のフリー・スペースが使用可能でなければなりません。 ファイルシステムは読み取り/書き込み用にマウントする必要があります。

ファイルシステムのパフォーマンスのチューニング

ファイルシステム・パフォーマンスのチューニングには多くの種類があります。

順次読み取りのパフォーマンスのチューニング

VMM 順次先読みフィーチャーによって、ラージ・ファイルの順次アクセスを行うプログラムのパフォーマンスを改善することができます。

VMM 順次先読みフィーチャーについては、 259 ページの『順次ページ先読み』に説明されています。

次の図は、典型的な先読み状態を示しています。

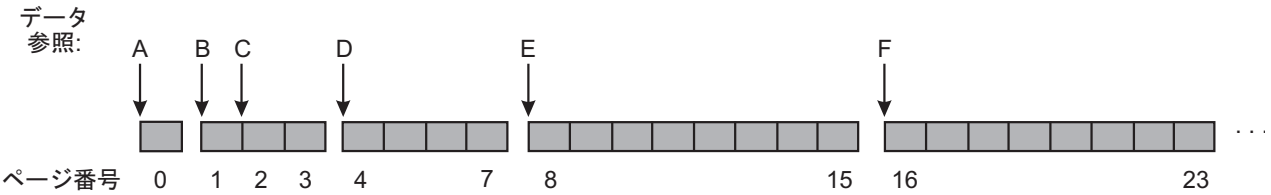


図 *20.* 順次先読みの例： この図に示したブロックの列は、連続したファイル・ページ番号のセグメントを表しています。これらのブロック・セグメントの番号は、0、1 から 3、4 から 7、8 から 15、16 から 23 です。 順次先読みの各ステップについては、この図のすぐ後の本文で説明しています。

この例では、*minpgahead* は 2 で、*maxpgahead* は 8 (デフォルト) です。 プログラムはファイルを順次に処理しています。先読みメカニズムにとって重要なデータ参照だけを、A から F によって示します。ステップの順序は次のとおりです。

A ファイルに対する最初のアクセスで、ファイルの最初のページ (ページ 0) が読み取られます。 この時点で、VMM はアクセスがランダムか順次かについて何も想定していません。

B プログラムが、間にファイルの他のページをアクセスせずに、次のページ (ページ 1) の最初のバイトをアクセスすると、VMM はプログラムが順次にアクセスしていると判断します。 そして、

minpgahead (2) の追加ページ (ページ 2 および 3) を読み取るようにスケジュールします。 したがって、アクセス B は合計 3 ページを読み取ることになります。

C プログラムが先読みされている最初のページ (ページ 2) の最初のバイトをアクセスすると、VMM は先読みページの値を倍の 4 にして、ページ 4 から 7 を読むようにスケジュールします。

D プログラムが先読みされている最初のページ (ページ 4) の最初のバイトをアクセスすると、VMM は先読みページの値を倍の 8 にして、ページ 8 から 15 を読むようにスケジュールします。

E プログラムが先読みされている最初のページ (ページ 8) の最初のバイトをアクセスすると、VMM は先読みページの値が *maxpgahead* に等しいと判断して、ページ 16 から 23 を読むようにスケジュールします。

F VMM は、プログラムが先読みページの前のグループの最初のバイトにアクセスすると、ファイルの終わりまで、*maxpgahead* ページの読み取りを続けます。

プログラムが順次アクセス・パターンから逸脱して、ファイルのページを順番にアクセスしないと、順次先読みは終了します。 VMM がプログラムが順次アクセスを再開したことを検出すると、順次先読みは *minpgahead* ページから再開されます。

minpgahead と *maxpgahead* の値は、**ioo** コマンドの **-r** および **-R** オプションを使用することによって変更できます。これらの値の変更を考えている場合は、次のことに注意してください。

- 値は 0、1、2、4、8、16、などの値の集合から選びます。他の値を使用すると、パフォーマンスまたは機能に逆効果となる可能性があります。
  - 値は、VMM が倍数アルゴリズムを使用しているので、2 の累乗でなければなりません。
  - *maxpgahead* の値が 16 より大きいと (64 KB を超える先読み)、一部のディスクのデバイス・ドライバーの能力を超えてしまいます。 このような場合、読み取りサイズは 64 KB に据え置かれます。
  - *maxpgahead* の値を高くすると、ストライピングされた論理ボリュームでの順次アクセスのパフォーマンスを最も重視するシステムで使用できます。
- *minpgahead* と *maxpgahead* の値がいずれも 0 の場合は、この機構が実質的に損なわれます。 これはパフォーマンスに対する逆効果になります。 ただし、入出力はランダムだが、入出力のサイズによって VMM の先読みアルゴリズムが有効になるような場合には、役に立つ場合があります。
- *maxpgahead* の値が 8 または 16 の場合、非ストライプ・ファイルシステムに対する順次入出力のパフォーマンスが最大になります。
- *minpgahead* から *maxpgahead* の先読み値の幅で十分高速なので、大部分のファイル・サイズでは、*minpgahead* を増やしても利点はありません。
- 順次先読みは、JFS と拡張 JFS では、別個に調整できます。 JFS では、ページ先読みは *minpgahead* と *maxpgahead* で調整できます。一方、拡張 JFS の場合は、*j2_minPageReadAhead* と *j2_maxPageReadAhead* が使用されます。

順次およびランダム遅延書き込みのパフォーマンスのチューニング

遅延書き込みでは、しきい値に達した後、**syncd** デーモンがページをディスクにフラッシュするのを待つのではなく、メモリー内の変更されたページをディスクに非同期に書き出します。

これは、メモリー内のダーティー・ページの数を制限し、システムのオーバーヘッドを減らし、またディスクのフラグメント化を最少にするために行われます。 遅延書き込みには、順次とランダムの 2 タイプがあります。

順次遅延書き込み:

クラスターの 4 ページのすべてがダーティーな場合、次のパーティションのページが変更されるとすぐに、このクラスターの 4 つのダーティー・ページはディスクへ書き出すようにスケジュールされます。 このフィーチャーがない場合は、ダーティー・ページが **syncd** デーモンが実行されるまでメモリーに残されるので、入出力ボトルネックやファイルのフラグメント化の原因となる可能性があります。

デフォルトでは、JFS ファイルは 16KB パーティション、すなわち 4 ページのパーティションに分割されます。これらの各パーティションはクラスター と呼ばれます。

VMM がしきい値として使用するクラスターの数はチューナブルです。 デフォルトは 1 クラスターです。 *numclust* パラメーターを **ioo -o** *numclust* コマンドを使用して増やすことによって、遅延書き込みを遅らせることができます。

拡張 JFS では、**ioo -o** *j2_nPagesPerWriteBehindCluster* コマンドは、一時にスケジュールされるページ数を指定するために使用されます。クラスター数を指定するのではありません。 拡張 JFS クラスターのページ数のデフォルトは 32 で、拡張 JFS のデフォルト・サイズが 128 KB であることを意味します。

ランダム遅延書き込み:

遅延書き込みフィーチャーは、ある特定のファイルのメモリー内のダーティー・ページの数が定義されたしきい値を超えると、その後に書き込まれるページをディスクに書き出すようにスケジュールするメカニズムを備えています。

多くのランダム入出力を行うアプリケーション、すなわち、入出力パターンが遅延書き込みアルゴリズムの要件に合わないために、 **syncd** デーモンが実行されるまで全ページがメモリーに常駐し続けるアプリケーションがあります。アプリケーションがメモリーの多くのページを変更した場合、 **syncd** デーモンが **sync()** コールを出すと、非常に多くのページがディスクに書き出される可能性があります。

ioo コマンドを JFS *maxrandwrt* パラメーター付きで使用することによって、しきい値をチューニングすることができます。デフォルト値は 0 で、これは、ランダム遅延書き込みが使用不可であることを示しています。 この値を 128 に増やすということは、いったんファイルの 128 個のメモリー常駐ページがダーティーになると、その後のダーティー・ページはいずれもディスクに書き出すようにスケジュールされるということを示します。最初のページの集合は、**sync()** コールの後にフラッシュされます。

拡張 JFS では、**ioo** コマンドのオプション **j2_nRandomCluster** (**-z** フラグ) および **j2_maxRandomWrite** (**-J** フラグ) がランダム遅延書き込みのチューニングに使用されます。 どちらのオプションも、デフォルトは 0 です。**j2_maxRandomWrite** オプションは、JFS に対する *maxrandwrt* の機能と同じ機能を拡張 JFS に対して持っています。 すなわち、このオプションは、メモリーに残ることができる、ファイル当たりのダーティー・ページ数の限度を指定します。 **j2_nRandomCluster** オプションは、2 つの連続する書き込みがランダムであると取られるために離れているべきクラスター数を指定します。

非同期ディスク入出力のパフォーマンスのチューニング

アプリケーションで同期入出力操作を行う場合、入出力が完了するのを待たなければなりません。 これに対して、非同期通信 I/O 操作はバックグラウンドで実行され、ユーザー・アプリケーションをブロックしません。 入出力操作とアプリケーション処理が同時に実行できるので、パフォーマンスが向上します。データベースやファイル・サーバーなどの多くのアプリケーションでは、処理と入出力をオーバーラップさせる能力を利用します。

アプリケーションでは、**aio_read()**、**aio_write()**、または **lio_listio()** サブルーチン (または対応する 64 ビットのサブルーチン) を使用して、非同期ディスク入出力を実行することができます。制御は、要求がキューされるとすぐにサブルーチンからアプリケーションに戻されます。すると、アプリケーションは、ディスク・オペレーションが実行されている間に処理を継続することができます。

非同期通信 I/O を管理するために、それぞれの非同期通信 I/O 要求には、アプリケーションのアドレス・スペースに対応する制御ブロックがあります。 この制御ブロックには、要求の制御と状況の情報が含まれています。 これは、入出力操作が完了したときに再び使用できます。

ユーザー・アプリケーションは、次の方法で、入出力操作が完了したときの通知方法を決めることができます。

- アプリケーションが、入出力操作の状況をポーリングする。
- システムが、入出力操作が行われたときに、非同期にアプリケーションに通知する。
- アプリケーションが、入出力操作が完了するまでブロックする。

各入出力は単一のカーネル・プロセスつまり kproc によって処理され、kproc は、一般にその入出力が完了するまで、キューからの他の要求を処理することはできません。 minservers チューナブルのデフォルト値は 3 で、maxservers のチューナブルのデフォルト値は 30 です。 maxservers 値は、プロセッサーごとの非同期入出力 kproc の数です。 AIX システムで実行されている非同期入出力 kproc の最大数を算出するには、maxservers 値に、現在実行されているプロセッサーの数を乗算します。

すべての AIO チューナブルには現行値、デフォルト値、最小値、および最大値があります。これらの値は、**ioo** コマンドを使用して表示することができます。 現行値のみが、**ioo** コマンドを使用して変更できます。 他の 3 つの値は固定されており、チューナブルの境界をユーザーに知らせるために提示されるものです。チューナブルの現行値はいつでも変更でき、オペレーティング・システムを再始動しても持続させることができます。 めったに非同期通信 I/O を使用するアプリケーションを実行しないシステムでは、通常デフォルトで十分です。

minservers と maxservers は両方ともプロセッサーごとのチューナブルですので注意してください。 これらのチューナブルは両方とも動的ですが、値を変更してもシステム内の使用可能サーバーの数が同期して変更されることはありません。 minservers の値が増加すると、並行入出力要求の数に直接比例して実際のサーバーの数が増加します。 新規の minservers 値に達すると、その値が新規フロアになります。 逆に、minservers が削減されると、非アクティビティーのためにサーバーが終了するので、使用可能サーバーの数はそのレベルまで自然に減少します。 非同期通信 I/O 要求の数が多い場合、maxservers を予測される同時入出力の数まで増やします。 AIO カーネル拡張は必要に応じて追加のサーバーを生成するので、minservers パラメーターは通常デフォルト値のままにしておくことをお勧めします。

注**:** ロー論理ボリュームまたは CIO モードでオープンされたファイルに対する AIO 入出力は、kproc サーバー・プロセスを使用しません。 maxservers と minservers の設定は、この場合には影響がありません。

AIO サーバーのプロセッサーの使用率を調べて、使用率がすべてのサーバーに均等に分割されている場合は、サーバーがすべて使用されていることを意味するので、この場合はサーバーの増加を検討することができます。 AIO サーバーを名前ごとに見るには、**pstat -a** コマンドを実行します。 AIO サーバーを kproc という名前で見るには、**ps -k** コマンドを実行します。

非同期ディスク入出力のパフォーマンスが重要で、要求の量が多いが、同時入出力のおおよその数が分からない環境では、 maxservers を少なくとも 10*(非同期にアクセスされるディスクの数) に設定することをお勧めします。

注**:** minservers または maxservers チューナブルへの変更を有効にするために、システムの再起動は必要ありません。 minservers チューナブル値は、平均ワークロードで最適のパフォーマンスを得られるレベルに設定する必要があります。

minservers チューナブルの値が maxservers チューナブルの値を超えることはできません。

非同期入出力チューニング値について詳しくは、『非同期入出力のチューニング値の変更』を参照してください。

ファイル同期のパフォーマンスのチューニング

ファイル同期化を拡張するには、幾つかの方法があります。

JFS ファイルの順次でない入出力は、次に示す一定の条件が成立するまでメモリーに累積されます。

- フリー・リストが *minfree* まで減少し、ページ置換が必要である。
- **syncd** デーモンが、定期的にスケジュールされた間隔でページをフラッシュする。
- **sync** コマンドが出される。
- ランダム遅延書き込みが、ランダム遅延書き込みのしきい値に達した後に、ダーティー・ページをフラッシュする。

上記のいずれかの条件が発生する前に多くのページが累積された場合は、**syncd** デーモンによってページがフラッシュされるときに、i ノード・ロックが取得され、すべてのダーティー・ページがディスクに書き出されるまで保持されます。 この間、そのファイルにアクセスしようとしているスレッドは、i ノード・ロックが使用不可であるためブロックされます。 **fuser** コマンドもファイルに対してブロックされます。これは、**fuser** では必要な情報を提供するために i ノード・ロックが必要であるからです。**syncd** デーモンは現在、ファイルのダーティー・ページをすべてフラッシュしますが、一時に 1 ファイルであることを覚えておいてください。 大量のメモリーを搭載していて多数のページが変更されているシステムでは、入出力のピークは、**syncd** デーモンがページをフラッシュするときに発生する可能性があります。

AIX には、**sync_release_ilock** と呼ばれるチューニング・オプションがあります。 **-o sync_release_ilock=1** オプション付きの **ioo** コマンドは、そのファイルのダーティー・ページのフラッシュ中に i ノード・ロックを解放できます。 この結果、**sync()** コール中にこのファイルをアクセスしている時の応答時間が良くなる場合があります。

このようなブロッキングの影響は、**syncd** デーモンでの同期の頻度を増やすことによって最小化することもできます。 **syncd** デーモンを開始する /sbin/rc.boot を変更します。 その後、システムをリブートして変更を有効にします。 現行システムの場合、**syncd** デーモンを kill し、新しい秒数値で再始動します。

この動作をチューニングする第三の方法は、**ioo** コマンドを使用してランダム遅延書き込みをオンにすることです (267 ページの『順次およびランダム遅延書き込みのパフォーマンスのチューニング』を参照)。

JFS2 同期チューナブル

ラージ・ファイルに対してランダムな入出力アクティビティーがある状況では、ファイルシステム同期操作は有効でない場合があります。 sync が発生すると、ユーザー・プログラムからファイルへのすべての読み取りと書き込みがブロックされます。ファイルに多くのダーティー・ページがある場合は、ディスクへの書き込みが終了するまでにかなりの時間を要します。 このような状況では、次の JFS2 チューナブル・パラメーターを使用することができます。

- j2_syncPageCount: 1 つのファイルに対して 1 回の sync で書き込まれる予定の変更ページ数を制限します。このチューナブルが設定されている場合、ファイルシステムは指定された数のページを書き込み、ファイルの残りの部分への入出力はブロックしません。 sync コールは、すべての変更ページが書き込まれるまで書き込み操作を繰り返します。
- j2_syncPageLimit: しきい値に達したときに、j2_syncPageCount パラメーターをオーバーライドします。 このパラメーターを使用すると、ファイルに対する sync 操作が確実に完了するようになります。これらのチューナブルは、標準の方法で保持されます。

チューナブルを操作するには、**ioo** コマンドを使用します。

j2_syncPageCount チューナブルと j2_syncPageLimit チューナブルは、**ioo** コマンドにより制御される値のリストに追加されます。

値を表示または変更するには –o フラグを使用し、ヘルプ・オプションを表示するには –h フラグを使用します。

```
# ioo –h j2_syncPageCount
```

sync システム・コールで 1 回の操作によりディスクに書き込まれる、1 つのファイルの変更ページの最大数を設定します。

値: デフォルト: 0 範囲: 0-65536

タイプ: 動的

単位: 4 KB ページ

チューニング: ファイルシステム・キャッシングを使用し、ランダムな書き込みを頻繁に行うアプリケーションを実行する場合は、同期操作中に起こる長時間のアプリケーション遅延を回避するようにこの設定を調整する必要があります。 この値は 256 から 1024 の範囲内でなければなりません。 デフォルト値はゼロです。この場合、1 回の呼び出しですべてのダーティー・ページが書き込まれる通常の同期動作が行われます。 チューナブルに小さい値が設定されている場合は、同期にかかる時間がより長くなり、アプリケーションの応答時間の遅延がより短くなります。 大きい値が設定されている場合は、応答時間の遅延がより長くなり、同期にかかる時間がより短くなります。

```
# ioo –h j2_syncPageLimit
```

sync システム・コールが j2_syncPageCount を使用する最大回数を設定し、書き込まれるページを制限して同期操作のパフォーマンスを向上させます。

値: デフォルト: 256 範囲: 16-65536

タイプ: 動的

単位: 数字

チューニング: j2_syncPageCount が設定されていて、j2_syncPageCount の変更による効果が十分でない場合にこれを増やす必要があるときに設定されます。 許容値は 250 から 8000 の範囲内です。 j2_syncPageCount が 0 である場合は、j2_syncPageLimit による効果はありません。

j2_syncPageCount が設定されていて、これを増やす必要がある場合は、j2_syncPageCount の変更による影響でアプリケーションの応答時間が損なわれないように、このチューナブルを設定する必要があります。

これらの値は 1 から 8000 の範囲内でなければなりません。 これらのチューナブルに最適な値は、メモリー・サイズと入出力の帯域幅により異なります。 中立的な開始点は、これらの両方のチューナブルを 256 に設定することです。

JFS2 の同期期間と並行処理

ファイルシステムの同期は、sync デーモン (**syncd**) によって管理されます。JFS2 チューナブル・パラメーターを使用すると、**syncd** を使用せずにファイルシステムが同期を処理できます。

JFS2 同期ハンドラーは、すべてのキャッシュ・データが同時にディスクに書き込まれないように、sync プロセスを分散します。sync は、一度に 1 つのファイルシステム上で実行されます。それぞれのファイルシステムは、前の操作が終了した後に次の sync 操作を開始するように設定されています。複数のファイルシステムを処理する必要がある場合に、sync 操作を処理するスレッドの数を増やすこともできます。

JFS2 ファイルシステムの同期に関しては、次のチューナブル・パラメーターを使用します。

j2_syncByVFS:
JFS2 同期ハンドラーの使用を指定し、ファイルシステムごとの sync 操作の間隔を設定します。

j2_syncConcurrency:
ファイルシステムの同期を処理するスレッドの数を設定します。この値は、sync 操作を並行して実行する必要があるファイルシステムの数を指定します。個々のファイルシステムに対して、ただ 1 つの sync スレッドが sync 操作を実行します。

ioo コマンドは、入出力のチューナブル・パラメーターを管理します。詳しくは、**ioo** コマンドの資料を参照してください。

```
# ioo –h j2_syncByVFS
```

目的**:** JFS2 ファイルシステムを同期するためのシステム呼び出しを反復する間に待機する秒数を指定します。この値は、**syncd** コマンドによって指定された値を置き換えます。

値**:** デフォルト: 0、範囲: 0 から 86400。

タイプ**:** 動的。

単位**:** 秒。

チューニング**:** この値は、sync プロセスを反復する間隔の秒数を指定します。 JFS2 同期ハンドラーは、JFS2 ファイルシステムに対する syncvfs サブルーチンの呼び出しを開始する前に、j2_syncByVFS チューナブル・パラメーターによって指定された時間まで待機します。値 0 は、通常の syncd 処理を使用するように指定します。ゼロ以外の値を指定すると、**syncd** コマンドで指定された時間がオーバーライドされ、JFS2 固有のファイル同期ハンドラーが使用されるようになります。

```
# ioo –h j2_syncConcurrency
```

目的**:** JFS2 sync 操作に使用されるスレッドの数を設定します。

値**:** デフォルト: 1、範囲: 1 から 128。

タイプ**:** 動的。

単位**:** 数値。

チューニング**:** sync デーモンは、j2_syncConcurrency チューナブル・パラメーターによって設定された数のファイルシステムに対して、sync 操作を並行して開始します。この値が有効になるのは、j2_syncByVFS チューナブル・パラメーターがゼロ以外の値である場合だけです。

関連情報:

ioo コマンド

ファイルシステム・バッファーのチューニング

ioo および **vmstat -v** の以下のパラメーターは、入出力バッファーのボトルネックの検出やディスク入出力のチューニングに役立ちます。

バッファー不足のためにブロックされた入出力のカウンター

vmstat -v コマンドは、さまざまなカーネル・コンポーネントでのバッファー不足のためにブロックされた入出力のカウンターを表示します。 以下に、**vmstat –v** 出力例の一部を示します。

```
...
              0 paging space I/Os blocked with no psbuf
              2740 filesystem I/Os blocked with no fsbuf
              0 external pager filesystem I/Os blocked with no fsbuf

...
```

「paging space I/Os blocked with no psbuf」および「filesystem I/Os blocked with no fsbuf」カウンターは、bufstruct が使用できず、VMM がスレッドを VMM 待ちリストに入れる度に増分されます。「external pager filesystem I/Os blocked with no fsbuf」カウンターは、拡張 JFS ファイルシステムで bufstruct が使用不可であった場合にその都度増分されます。

numfsbufs パラメーター

ファイルシステムに対して多くの同時入出力または大量の入出力がある場合、またはファイルシステムに対する大量の順次入出力がある場合、bufstructs を待っている間に、ファイルシステム・レベルで入出力がボトルネックになる可能性があります。 ファイルシステム当たりの bufstructs の数 (*numfsbufs* と呼ばれる) は、**ioo** コマンドを使用して増やすことができます。値が有効になるのは、ファイルシステムがマウントされるときだけです。そのため、値を変更する場合は、ファイルシステムをアンマウントしてから再びマウントする必要があります。 *numfsbufs* のデフォルト値は現在、ファイルシステム当たり 93 bufstructs です。

j2_nBufferPerPagerDevice パラメーター

注**:** vmstat -v が拡張 JFS のファイルシステムでの buffstructs の不足を示すとき、*j2_dynamicBufferPreallocation* 調整可能値は、 *j2_nBufferPerPagerDevice* パラメーターにどのような変更も加える前に、最初に調整する必要があります。

拡張 JFS では、bufstructs 数は、*j2_nBufferPerPagerDevice* パラメーターで指定されます。拡張 JFS ファイルシステムにおける bufstructs 数のデフォルトは、現在は 512 です。 拡張 JFS ファイルシステム当たりの bufstructs 数 (*j2_nBufferPerPagerDevice*) は、**ioo** コマンドを使用して増すことができます。 値が有効になるのは、ファイルシステムがマウントされるときだけです。

lvm_bufcnt パラメーター

アプリケーションがファイルシステムを通して書き込むのでなく、非常に大量のロー入出力を出している場合、ファイルシステムと同タイプのボトルネックが LVM レイヤーで起こる可能性があります。 LVM レイヤーでボトルネックが起こるのは、非常に大量の入出力を非常に高速の入出力装置に対して行った場合で

す。 しかし、そうなった場合、*lvm_bufcnt* と呼ばれるパラメーターを **ioo** コマンドによって増やし、多数の「uphysio」バッファーを提供することができます。 値は即時に有効になります。 「uphysio」バッファーの現在のデフォルト値は 9 です。 LVM は現在入出力をそれぞれ 128 K に分割しているので、*lvm_bufcnt* のデフォルト値は 9 で、一度に 9*128 K を書き込むことができます。 入出力が 9*128 K より大きいときは、*lvm_bufcnt* を増やすと効果がある場合があります。

pd_npages パラメーター

pd_npages パラメーターは、ファイルを削除するときに、RAM から 1 つのチャンク (大きい塊) で削除するページの数を指定します。 この値を変更して有利になるのは、ファイルを削除するリアルタイム・アプリケーションだけです。 *pd_npages* パラメーターの値を減らせば、リアルタイム・アプリケーションは、プロセス/スレッドがディスパッチされる前に削除されるページ数が少なくなるので、応答時間が良くなる可能性があります。 デフォルト値は可能な最大ファイル・サイズをページ・サイズ (現在 4096) で割った値です。可能な最大ファイル・サイズが 2 GB の場合、*pd_npages* パラメーターの値はデフォルトによって 524288 になります。

v_pinshm パラメーター

v_pinshm パラメーターを 1 に設定すると、共有メモリー・セグメント内のページが VMM によって固定されます。これは、**shmget()** を実行するアプリケーションが、フラグの一部として SHM_PIN を指定する場合です。 デフォルト値は 0 です。

アプリケーションでは、そのアプリケーションが SHM_PIN フラグを使用するべきかどうかを指定するチューニング項目を選択することができます (例えば、Oracle 8.1.5 以降の *lock_sga* パラメーター)。 大量のメモリーを固定するのは避けてください。そのような場合、ページ置換が起こらなくなるからです。 固定が役に立つのは、これらの共有メモリー・セグメントで非同期通信 I/O のオーバーヘッドが節約できるからです (非同期通信 I/O カーネル・エクステンションは、バッファーの固定には必要ありません)。

直接入出力のチューニング

直接入出力の主な利点は、VMM ファイル・キャッシュからユーザー・バッファーへのコピーを除くことによって、ファイルの読み取りおよび書き込みの CPU 使用率を低減することができます。

ファイルに対する通常入出力を処理している場合、入出力はアプリケーション・バッファーから VMM へ、そしてそこからアプリケーション・バッファーに戻ります。 バッファーの内容は、VMM が実メモリーをファイル・バッファー・キャッシュとして使用することによって、RAM にキャッシュされます。 ファイルのキャッシュ・ヒット率が高い場合、このタイプのキャッシュ入出力は非常に効果的で、入出力全体のパフォーマンスが改善されます。 しかし、キャッシュ・ヒット率が低いアプリケーションや非常に大量の入出力を行うアプリケーションでは、通常のキャッシュ入出力を使用すると、それほど利益を得られない場合があります。

キャッシュ・ヒット率が低ければ、大部分の読み取り要求はディスクに行く必要があります。 大部分の場合、書き込みは通常のキャッシュ入出力の方が速くなります。 しかし、ファイルが O_SYNC または O_DSYNC でオープンされた場合は (229 ページの『sync および fsync コール』を参照)、ディスクに書き出す必要があります。 このような場合は、データ・コピーが除去されるので、直接入出力の方がアプリケーションに有利になります。

もう 1 つの利点は、直接入出力では、アプリケーションが他のファイルのキャッシュの有効性を弱めてしまうのを避けられるということです。 ファイルの読み取りまたは書き込み時に、そのファイルはメモリー内でスペースの競合が発生し、その結果、他のファイル・データがメモリーから押し出されることがありま

す。 アプリケーション開発者が、あるファイルのキャッシュ使用率の特性が低いことを知っていれば、それらのファイルだけを O_DIRECT でオープンすることができます。

直接入出力を効率的に機能させるには、入出力要求を、使用するファイルシステムのタイプに適したものにする必要があります。 **finfo()** および **ffinfo()** サブルーチンを使用して、固定ブロック・サイズのファイルシステム、フラグメント化されたファイルシステム、およびビッグ・ファイルのファイルシステムのそれぞれについて、オフセット、長さ、およびアドレス位置合わせ要件を照会できます (直接入出力は、圧縮されたファイルシステムではサポートされていません)。 照会される情報は、`/usr/include/sys/finfo.h` に記述されているように、構造体 **diocapbuf** の中に格納されています。

整合性の問題を避けるために、ファイルのオープンの呼び出しが複数あり、その中の 1 つ以上で O_DIRECT が指定されておらず、別のオープンでは O_DIRECT が指定されている場合、ファイルは通常のキャッシュ入出力モードのままです。 同様に、**shmat()** または **mmap()** システム・コールによってメモリーにマップされているファイルも、通常のキャッシュ・モードのままになります。 最後の競合する、非直接アクセスが除去されると、ファイルシステムはファイルを直接入出力モードに移動します (**close()**、**munmap()**、または **shmdt()** のいずれかのサブルーチンを使用して)。通常モードから直接入出力モードへの移動は、メモリー内の変更されたページをすべてその時点でディスクにフラッシュしなければならないので、コストがかかります。

直接入出力に必要な CPU サイクルは、通常の入出力より大幅に少ないので、通常入出力の提供するキャッシングの利点をあまり生かせない入出力集中のアプリケーションでは、直接入出力を使用することで、パフォーマンスを改善することができます。CPU のスピードの向上がメモリー・スピードの向上より優っているので、直接入出力による利益は今後もさらに大きくなっていきます。

直接入出力の一般的な候補は、CPU 制約で、大量のディスク入出力を実行するプログラムです。大量の順次入出力のあるテクニカル・アプリケーションは、優れた候補です。 多数の小さい入出力を行うアプリケーションでは、一般にパフォーマンス上の利点は少なくなります。直接入出力では先読みや遅延書き込みが行われないからです。 ストライピングによって利益を得るアプリケーションも優れた候補です。

直接入出力の読み取りパフォーマンス:

直接入出力の使用によって CPU 使用率を低減できるとしても、一般に経過時間が (特に小さい入出力要求では) 長くなります。要求がメモリーにキャッシュされないからです。

直接入出力読み取りはディスクからの同期読み取りを発生させますが、通常のキャッシュ・ポリシーでは、キャッシュから読み取ります。 この結果、通常のキャッシュ・ポリシーではデータがメモリーに入る傾向があった場合、パフォーマンスが低下します。 さらに直接入出力では、入出力が VMM を通らないので、VMM 先読みアルゴリズムをバイパスします。 先読みアルゴリズムでは、VMM がディスク要求を開始して、アプリケーションが要求する前にページをメモリーに常駐させることができるので、ファイルの順次アクセスには非常に役に立ちます。 アプリケーションでは、次の方式の 1 つを使用することによって、先読みでの損失を埋め合わせることができます。

- 大きい読み取り要求 (最低 128 K) を出す。
- 複数のスレッドを使用することによって、非同期の直接入出力先読みを出す。
- **aio_read()** や **lio_listio()** などの、非同期通信 I/O 機能を使用する。

直接入出力の書き込みパフォーマンス:

直接入出力書き込みでは、VMM をバイパスして直接ディスクに書き出すので、重大なパフォーマンスのペナルティーを伴う場合があります。通常のキャッシュ入出力では、書き込みをメモリーに入れてから、非同期 または遅延書き込み 操作によって後でディスクにフラッシュすることができます。

直接入出力書き込みではメモリーにコピーを取らないので、同期操作を実行するときに、これらのページをディスクにフラッシュする必要がありません。したがって、**syncd** デーモンの実行に必要な作業量を低減することができます。

ファイルシステム・ログおよびログ論理ボリュームの再編成

ジャーナル・ファイルシステム (JFS) および拡張ジャーナル・ファイルシステム (JFS2) では、データベース・ジャーナリング手法を使用して、整合性のあるファイルシステム構造を維持しています。 これには、ファイルシステム・メタデータに対して行われたトランザクションを循環式のファイルシステム・ログに複写することも含まれます。 ファイルシステム・メタデータには、スーパーブロック、i ノード、間接データ・ポインター、およびディレクトリーが含まれます。

メモリー内のページが **sync()** コールまたは **fsync()** コールによって実際にディスクに書き出されると、データが現在ディスク上にあることを示すために、コミット・レコードがログに書き込まれます。 ログのトランザクションは、次のような状況で発生します。

- ファイルが作成または削除されたとき。
- O_SYNC によってオープンされたファイルに対して **write()** コールが発生し、書き込みによってディスク・ブロックの新規割り当てが行われたとき。
- **fsync()** または **sync()** サブルーチンが呼び出されたとき。
- 書き込みによって、間接ブロックまたは二重間接ブロックが割り当てられることになったとき。

ファイルシステム・ログを使用すると、システムがダウンした場合に、迅速かつクリーンなファイルシステムのリカバリーが可能になります。 アプリケーションで同期入出力を行っている場合、あるいは多くのファイルを短時間に作成または除去している場合、ログの論理ボリュームで大量の入出力が発生している可能性があります。 ログの論理ボリュームとファイルシステム論理ボリュームが両方とも同じ物理ディスク上にある場合、これが入出力のボトルネックになる可能性があります。お勧めするのは、ロギング用装置を別の物理ディスクに移すことです (これは、NFS サーバーの場合に特に役に立ちます)。

高速書き込みキャッシュ・デバイスを使用すると、ログ論理ボリューム (ファイルシステム・ログまたはデータベース・ログ) で非常に良いパフォーマンスを得ることができます。

AIX には、JFS ファイルシステム用に **nointegrity** と呼ばれる **mount** オプションが用意されており、そのオプションを指定してマウントされたファイルシステムで JFS ログの使用をバイパスします。 システムがクリーンなシャットダウンを行わずにダウンした場合は、アドミニストレーターが **fsck** コマンドの実行が必要な場合があるということを知っている限りは、この方法を使用するとパフォーマンスが良くなる可能性があります。

ファイルシステム・ログに対する入出力を記録するには、**filemon** コマンドを使用します。 ファイルシステムとそのロギング用装置が両方とも過度に利用されているようであれば、それぞれを別の物理ディスクに置いた方が良い場合があります (そのボリューム・グループに複数のディスクがあるとして)。

1 つのボリューム・グループに複数のロギング用装置を入れることができます。 ただし、ファイルシステムのログは、そのファイルシステムと同じボリューム・グループに入れる必要があります。ログ論理ボリュームまたはファイルシステム論理ボリュームは、システムが稼働中で使用中であっても、**migratepv** コマンドを使用して、別のディスクに移動することができます。

関連概念:

223 ページの『ボリューム・グループに関する推奨事項』
システム管理を容易にし、パフォーマンスを良くするために、可能なら、オペレーティング・システムが最初にインストールされる物理ボリュームだけでデフォルト・ボリューム・グループ (rootvg) を構成しま

す。

ログ論理ボリュームの作成

ログ論理ボリュームを、最もアクティブなファイルシステムの物理ボリュームと異なる物理ボリューム上に置くと、リソースの並列使用が増加します。 各ファイルシステムごとに別のログを使用することができます。

論理ボリュームを作成すると、ドライブのパフォーマンスが異なります。 ホットなファイルシステムの論理ボリュームは、次のようにして、高速ドライブ (可能なら、高速の書き込みキャッシュを持つもの) 上に作成するようにしてください。

1. 次のようにして、新規ファイルシステムのログ論理ボリュームを作成します。

   ```
   # mklv -t jfslog -y LVname VGname 1 PVname
   ```

   または

   ```
   # mklv -t jfs2log -y LVname VGname 1 PVname
   ```

   または

   ```
   # smitty mklv
   ```

2. 次のように、ログをフォーマットします。

   ```
   # /usr/sbin/logform -V vfstype /dev/LVname
   ```

3. /etc/filesystems と論理ボリューム制御ブロック (LVCB) を、次のように変更します。

   ```
   # chfs -a log=/dev/LVname /filesystemname
   ```

4. ファイルシステムをアンマウントしてからマウントします。

独立のボリューム上にログを作成する別の方法は、次のとおりです。

- 最初に、単一物理ボリュームを持つボリューム・グループを定義します。
- 新規ボリューム・グループ内に論理ボリュームを定義します (この結果、ボリューム・グループ JFS ログの割り当てが最初の物理ボリューム上になります)。
- 残りの物理ボリュームをボリューム・グループに追加します。
- 使用頻度が高いファイルシステム (論理ボリューム) は、新たに追加された物理ボリューム上に定義します。

ディスク入出力ペーシング

ディスク入出力ペーシングは、非常に大量の出力を生成するプログラムによってシステムの入出力機能が飽和状態になり、要求の少ないプログラムの応答時間が低下するのを避けることを目的としています。

ディスク入出力ペーシングでは、すべての保留入出力の合計について、セグメント単位、つまりファイル単位ごとの最高水準点と最低水準点を設定します。 プロセスが、既に最高水準点 の保留書き込みのあるファイルに書き込もうとすると、そのプロセスは、保留書き込みの数が最低水準点 以下になるのに十分な入出力が完了するまで、スリープ状態に入ります。 入出力要求を処理する論理は変わりません。 大容量のプロセスからの出力は多少スローダウンします。

最高水準点と最低水準点は、SMIT ツールでシステム全体に設定できます。これを行うには、**maxpout** および **minpout** マウント・オプションを使用して、「システム環境」**->**「オペレーティング・システムの特性の変更/表示」 (**smitty chgsys**) を選択してから、最高水準点と最低水準点に対するページ数、または個々のファイルシステムに対するページ数を入力します。

maxpout パラメーターは、スレッドの中断前にファイルに対して入出力状態でスケジュール可能なページ数を指定します。 **minpout** パラメーターでは、スケジュールされる最小ページ数を指定して、そのページ数の状態になるとスレッドが中断状態からウェイクアップします。 **maxpout** のデフォルト値は 8193、**minpout** は 4096 です。 入出力ペーシングを使用不可にするには、単に両方の値をゼロに設定します。

パラメーター **maxpout** および **minpout** のシステム全体値への変更は、システムをリブートせずに、すぐに有効になります。 パラメーター **maxpout** および **minpout** の値を変更すると、システム全体の設定が上書きされます。 ファイルシステムをマウントし、パラメーター **maxpout** および **minpout** の値を明示的に 0 に設定することにより、システム全体の入出力ペーシングからファイルシステムを除外することができます。以下のコマンドはその一例です。

```
mount -o minpout=0,maxpout=0 /<file system>
```

パラメーター **maxpout** および **minpout** をチューニングすると、ファイルに順次書き込みを行っているどのスレッドもシステム・リソースを占有しないようにすることができます。

下表に、IBM eServer™ pSeries モデル 7039-651 上での **vi** エディター・セッションの応答時間を例示します。このシステムは、1.7 GHz プロセッサーを装備した 4-way システムとして構成され、ディスク書き込み時のパラメーター **maxpout** および **minpout** に各種の値を指定しています。

maxpout の値	**minpout** の値	**dd** ブロック・サイズ **(10 GB)**	書き込み **(秒)**	スループット **(MB/秒)**	**vi** コメント
0	0	10000	201	49.8	dd 完了後
33	24	10000	420	23.8	遅延なし
65	32	10000	291	34.4	遅延なし
129	32	10000	312	32.1	遅延なし
129	64	10000	266	37.6	遅延なし
257	32	10000	316	31.6	遅延なし
257	64	10000	341	29.3	遅延なし
257	128	10000	223	44.8	遅延なし
513	32	10000	240	41.7	遅延なし
513	64	10000	237	42.2	遅延なし
513	128	10000	220	45.5	遅延なし
513	**256**	**10000**	**206**	**48.5**	遅延なし
513	384	10000	206	48.5	3 から 6 秒
769	512	10000	203	49.3	15 から 40 秒。これより長くなる可能性あり。
769	640	10000	207	48.3	3 秒未満
1025	32	10000	224	44.6	遅延なし
1025	64	10000	214	46.7	遅延なし
1025	128	10000	209	47.8	1 秒未満
1025	256	10000	204	49.0	1 秒未満
1025	384	10000	203	49.3	3 秒
1025	512	10000	203	49.3	25 から 40 秒。これより長くなる可能性あり。
1025	640	10000	202	49.5	7 から 20 秒。これより長くなる可能性あり。
1025	768	10000	202	49.5	15 から 95 秒。これより長くなる可能性あり。

maxpout の値	minpout の値	dd ブロック・サイズ (10 GB)	書き込み (秒)	スループット (MB/秒)	vi コメント
1025	896	10000	209	47.8	3 から 10 秒

パラメーター **maxpout** および **minpout** の最適範囲は、CPU 速度と入出力システムによって異なります。 I/O ペーシングは、**maxpout** パラメーターの値が **j2_nPagesPerWriteBehindCluster** パラメーターの値と等しいかそれ以上であると、十分に機能します。例えば、**maxpout** パラメーターの値が 64 で **minpout** パラメーターの値が 32 の場合、入出力状態で多くても 64 ページあり、2 回の入出力があってから、次の書き込みでブロッキングが発生します。

デフォルトのチューニング・パラメーターは、次のとおりです。

パラメーター	デフォルト値
j2_nPagesPerWriteBehindCluster	32
j2_nBufferPerPagerDevice	512

拡張 JFS では、**ioo -o j2_nPagesPerWriteBehindCluster** コマンドを使用して、一度にスケジュールされるページ数を指定できます。 拡張 JFS クラスターのページ数のデフォルトは 32 で、これは、拡張 JFS のデフォルト・サイズが 128 KB であることを意味します。 **ioo -o j2_nBufferPerPagerDevice** コマンドを使用して、ファイルシステム bufstruct 数を指定できます。 デフォルト値は 512 です。 この値を有効にするには、ファイルシステムを再マウントする必要があります。

拡張 JFS では、マウント用の **-o remount** コマンドを使用して、既にマウント済みファイルシステムの **maxpout** 値および **minpout** 値を変更できます。

ネットワーク・パフォーマンス

AIX は、モニターおよびチューニングを行うツールとメソッドだけでなく、各種通信プロトコルを提供します。

TCP および UDP のパフォーマンスのチューニング

チューナブルな通信パラメーターの最適な設定は、機能に優れたシステムやアプリケーション・プログラムの通信入出力特性と同様に、LAN のタイプによって異なります。 このセクションでは、AIX のための通信のチューニングの全体的な原則について説明します。

ネットワークのインストールとワークロードに関する検査とチューニングの概略は、以下のとおりです。

- アダプターが正しいスロットに配置されていることを確認します。
- システム・ファームウェアが正しいリリース・レベルであることを確認します。
- アダプター・スイッチとネットワーク・スイッチが正しい速度で、かつ二重モードであることを確認します。
- 正しい MTU サイズが選択されていることを確認します。
- ネットワークのタイプ、速度、プロトコルのための AIX チューニング・オプションを調整します。
- その他の考慮事項:
  - アダプター・オフロードのオプション
    - TCP チェックサム・オフロード
    - TCP 大容量送信または再セグメンテーション

– 割り込みの合体
– 入力スレッド (ドッグ・スレッド)

アダプターの配置

ネットワークのパフォーマンスは、選択するハードウェア (アダプターのタイプ) やマシン内のアダプターの配置に依存します。

最高のパフォーマンスを確保するために、ネットワーク・アダプターを各アダプターに最適な入出力バス・スロットに配置してください。

いずれの入出力バス・スロットが最適であるかを判別するには、以下の要因を検討してください。

- PCI-X アダプターと PCI アダプターの比較
- 64 ビット・アダプターと 32 ビット・アダプターの比較
- サポートされているバス・スロット・クロック速度 (33 MHz、50/66 MHz、または 133 MHz)

アダプターの帯域幅やデータ転送速度が高くなればなるほど、スロットの配置が重要になります。 例えば、PCI-X アダプターは、一般的にバス上で 133 MHz クロック速度で実行されるので、PCI-X スロットで使用した場合に最高のパフォーマンスを発揮します。 PCI-X アダプターを PCI スロットに配置することも可能ですが、通常はバス上で 33 MHz または 66 MHz で実行されるので速度が落ちます。また、いくつかのワークロードではうまく動作しません。

同様に、64 ビット・アダプターは、64 ビット・スロットにインストールした場合に最高の働きをします。64 ビット・アダプターを 32 ビット・スロットに配置することも可能ですが、最適な速度で動作しません。 ジャンボ・フレーム・モードでのギガビット・イーサネットのような大容量 MTU アダプターは、64 ビット・スロットで使用するほうがはるかにうまく動作します。

パフォーマンスに影響を与える可能性がある他の問題としては、バスまたは PCI ホスト・ブリッジ (PHB) ごとのアダプターの数です。 システム・モデルおよびアダプター・タイプによっては、 PHB ごとの高速アダプターの数は制限される可能性があります。 配置のガイドラインによって、アダプターをさまざまな PCI バス上に分散し、必要に応じて PCI バスごとのアダプターの数を制限してください。 マシン・モデルとアダプター・タイプごとの詳細については、*PCI* アダプターの配置リファレンス を参照してください。

IBM System p マシンで使用可能な PCI スロットと PCI-X スロットを以下の表にまとめます。

スロット・タイプ	このトピックで使用するコード
PCI 32 ビット 33 MHz	A
PCI 32 ビット 50/66 MHz	B
PCI 64 ビット 33 MHz	C
PCI 64 ビット 50/66 MHz	D
PCI-X 32 ビット 33 MHz	E
PCI-X 32 ビット 66 MHz	F
PCI-X 64 ビット 33 MHz	G
PCI-X 64 ビット 66 MHz	H
PCI-X 64 ビット 133 MHz	I

より新しい IBM Power Systems プロセッサー・ベースのサーバーにのみ、PCI-X スロットがあります。PCI-X スロットには、 PCI アダプターとの下位互換性があります。

一般的なアダプターの例と推奨スロット・タイプを以下の表にまとめます。

アダプター・タイプ	優先スロット・タイプ (最低から最高の優先順位)
10/100 Mbps イーサネット PCI アダプター II (10/100 イーサネット) FC 4962	A から I
IBM PCI 155 Mbps ATM アダプター FC 4953 または 4957	D、H、I
IBM PCI 622 Mbps MMF ATM アダプター FC 2946	D、G、H、I
Gigabit Ethernet-SX PCI アダプター FC 2969	D、G、H、I
IBM 10/100/1000 Base-T イーサネット PCI アダプター FC 2975	D、G、H、I
Gigabit Ethernet-SX PCI-X アダプター (ギガビット・イーサネット・ファイバー)、FC 5700	G、H、I
10/100/1000 Base-TX PCI-X アダプター (ギガビット・イーサネット) FC 5701	G、H、I
2-ポート Gigabit Ethernet-SX PCI-X アダプター (ギガビット・イーサネット・ファイバー)、FC 5707	G、H、I
2-ポート 10/100/1000 Base-TX PCI-X アダプター (ギガビット・イーサネット) FC 5706	G、H、I
10 Gigabit-SR Ethernet PCI-X アダプター、FC 5718	I (PCI-X 133 スロットのみ)
10 Gigabit-LR Ethernet PCI-X アダプター、FC 5719	I (PCI-X 133 スロットのみ)

lsslot -c pci コマンドは、以下の情報を提供します。

- スロットの PCI タイプ
- バス速度
- デバイスとスロットの関係

6 個の内部スロットを備えた 2-way p615 システム上での **lsslot -c pci** コマンドの例を以下に示します。

```
# lsslot -c pci
# Slot      Description                       Device(s)
U0.1-P1-I1  PCI-X capable, 64 bit, 133 MHz slot  fcs0
U0.1-P1-I2  PCI-X capable, 32 bit, 66 MHz slot   Empty
U0.1-P1-I3  PCI-X capable, 32 bit, 66 MHz slot   Empty
U0.1-P1-I4  PCI-X capable, 64 bit, 133 MHz slot  fcs1
U0.1-P1-I5  PCI-X capable, 64 bit, 133 MHz slot  ent0
U0.1-P1-I6  PCI-X capable, 64 bit, 133 MHz slot  ent2
```

ギガビット・イーサネット・アダプターの場合、**entstat -d en***[interface-number]* コマンド出力または **netstat -v** コマンド出力の末尾に、アダプター特有の統計情報として PCI バスのタイプやアダプターのバス速度が表示されます。 以下に、 **netstat -v** コマンドの出力例を示します。

```
# netstat -v

10/100/1000 Base-TX PCI-X Adapter (14106902) Specific Statistics:
--------------------------------------------------------------------
Link Status: Up
Media Speed Selected: Auto negotiation
Media Speed Running: 1000 Mbps Full Duplex
PCI Mode: PCI-X (100-133)
PCI Bus Width: 64 bit
```

システム・ファームウェア

システム・ファームウェアは、システムのさまざまな入出力バスや PCI バス上の入出力チップのオプションを構成する役割のほかに、各 PCI アダプターのキー・パラメーターを構成する役割も果たします。

いくつかのケースでは、ファームウェアは個々のアダプターに特有なパラメーターを設定します。例えば、PCI 待ち時間タイマーや、キャッシュ・ライン・サイズ、PCI-X アダプターの最大メモリー読み込みバイト数 (MMRBC) の値などです。 これらのパラメーターは、アダプターのパフォーマンスを良くするために重要です。 ファームウェアが下位レベルであるためにこれらのパラメーターが適切に設定されていない場合、ソフトウェアのチューニングだけで最適なパフォーマンスを実現することは不可能です。 新しいアダプターをシステムへ追加する前に、古いシステムのファームウェアを必ず更新してください。

以下の例のとおり、プラットフォーム・ファームウェアとシステム・ファームウェアのレベルは、**lscfg -vp|grep -p " ROM"** コマンドで確認できます。

```
lscfg -vp|grep -p " ROM"

     ...lines omitted...

     System Firmware:
        ROM Level (alterable).......M2P030828
        Version.....................RS6K
        System Info Specific.(YL)...U0.1-P1/Y1
      Physical Location: U0.1-P1/Y1

      SPCN firmware:
        ROM Level (alterable).......0000CMD02252
        Version.....................RS6K
        System Info Specific.(YL)...U0.1-P1/Y3
      Physical Location: U0.1-P1/Y3

      SPCN firmware:
        ROM Level (alterable).......0000CMD02252
        Version.....................RS6K
        System Info Specific.(YL)...U0.2-P1/Y3
      Physical Location: U0.2-P1/Y3

      Platform Firmware:
        ROM Level (alterable).......MM030829
        Version.....................RS6K
        System Info Specific.(YL)...U0.1-P1/Y2
      Physical Location: U0.1-P1/Y2
```

アダプターのパフォーマンスのガイドライン

AIX オペレーティング・システムは、アダプター・パフォーマンスを最大化するためのいくつかのガイドラインを提供しています。

ユーザー・ペイロード・データ転送速度を確認するには、TCP 接続でデータをストリーミングするアプリケーション用のソケット・ベースのプログラムを使用します。 例えば、1 つのプログラムが **send( )** コールを実行し、受信側が **recv( )** コールを実行します。転送速度は、ネットワーク・ビット・レート、MTU サイズ (フレーム・サイズ)、物理レベル・オーバーヘッド (フレーム間ギャップやプリアンブル・ビットなど)、データ・リンク・ヘッダー、および TCP/IP ヘッダーと、ギガヘルツ速度のプロセッサーの相関関係によって決まります。以下の転送速度は、単一 LAN の場合の最高の速度であって、ルーターや追加ネットワーク・ホップやリモート・リンクを通れば速度が落ちます。

片方向 (simplex) TCP ストリーミング速度は、メモリー間テストで、システム A からシステム B への FTP データ送信テストのようなワークロードによって確認できる速度です。 323 ページの『ftp コマンド』を参照してください。全二重通信メディアは、TCP 肯定応答が、データ・パケットが流れる同じワイヤー上を競合なしでフロー・バックできるので、半二重通信メディアより速くなります。

注**:** 以下の表の「ロー・ビット・レート」値は、物理メディア・ビット・レートであり、フレーム間ギャップ、プリアンブル・ビット、セル・オーバーヘッド (ATM の場合)、データ・リンク・ヘッダー、トレーラーなどの物理メディア・オーバーヘッドを反映しません。これらの値は、ワイヤーの有効使用可能ビット・レートを落とします。

最大ネットワーク・ペイロード速度と片方向 (simplex) TCP ストリーミング速度を次の表に示します。

表 *5.* 片方向通信 *TCP* ストリーミング速度に対する最大ネットワーク・ペイロード速度

ネットワーク・タイプ	ロー・ビット・レート **(M** ビット**)**	ペイロード・レート **(Mb)**	ペイロード・レート **(MB)**
10 Mb イーサネット、半二重	10	6	0.7
10 Mb イーサネット、全二重	10 (20 Mb 全二重)	9.48	1.13
100 Mb イーサネット、半二重	100	62	7.3
100 Mb イーサネット、全二重	100 (200 Mb 全二重)	94.8	11.3
1000 Mb イーサネット、全二重、MTU 1500	1000 (2000 Mb 全二重)	948	113.0
1000 Mb イーサネット、全二重、MTU 9000	1000 (2000 Mb 全二重)	989	117.9
10 Gb イーサネット、全二重、MTU 1500 (RFC1323 使用可能)	10000	7200 (ピーク 9415)[1]	858 (ピーク 1122)[1]
10 Gb イーサネット、全二重、MTU 9000 (RFC1323 使用可能)	10000	9631 (ピーク 9891)[1]	1148 (ピーク 1179)[1]
FDDI、MTU 4352 (デフォルト)	100	92	11.0
非同期伝送モード (ATM) 155、MTU 1500	155	125	14.9
ATM 155、MTU 9180 (デフォルト)	155	133	15.9
ATM 622、MTU 1500	622	364	43.4
ATM 622、MTU 9180 (デフォルト)	622	534	63.6

[1] 表に示す値は、専用パーティションで専用アダプターの場合の速度です。仮想イーサネット・アダプター (VIOS 内) または共用イーサネット・アダプター (SEA) の場合、または共用区画 (共用 LPAR) の場合の 10 ギガビット・イーサネット・アダプターのパフォーマンスは、この表には示しません。パフォーマンスは他の変数およびチューニングに影響されますが、それらはこの表の範囲外です。

双方向 (duplex (二重)) TCP ストリーミング・ワークロードには、両方向のデータ・ストリーミングがあります。 例えば、システム A からシステム B に **ftp** コマンドを実行し、別の **ftp** コマンド・インスタンスをシステム B からシステム A に同時に実行することは、二重 TCP ストリーミングと見なされます。この種のワークロードは、データの送信と受信を同時に実行できる全二重メディアの利点を活用しています。 半二重モードの光ファイバー分散データ・インターフェース (FDDI) やイーサネットのような一部のメディアは、データの送信と受信を同時に実行することができないため、二重通信ワークロードを実行する場合にうまく動作しません。二重通信ワークロードは片方向通信ワークロードの 2 倍の速度になるわけではありません。受信側から戻ってくる TCP 肯定応答パケットが、同一方向のデータ・パケットの流れと競合するからです。双方向 (duplex (二重)) TCP ストリーミング速度を以下の表にまとめます。

表 6. 二重 *TCP* ストリーミング速度に対する最大ネットワーク・ペイロード速度

ネットワーク・タイプ	ロー・ビット・レート **(M** ビット**)**	ペイロード・レート **(Mb)**	ペイロード・レート **(MB)**
10 Mb イーサネット、半二重	10	5.8	0.7
10 Mb イーサネット、全二重	10 (20 Mb 全二重)	18	2.2
100 Mb イーサネット、半二重	100	58	7.0
100 Mb イーサネット、全二重	100 (200 Mb 全二重)	177	21.1
1000 Mb イーサネット、全二重、MTU 1500	1000 (2000 Mb 全二重)	1811 (1667 ピーク) [1]	215 (222 ピーク) [1]
1000 Mb イーサネット、全二重、MTU 9000	1000 (2000 Mb 全二重)	1936 (1938 ピーク) [1]	231 (231 ピーク) [1]
10 Gb イーサネット、全二重、MTU 1500	10000 (20000 Mb 全二重)	14400 (18448 ピーク) [1]	1716 (2200 ピーク) [1]
10 Gb イーサネット、全二重、MTU 9000	10000 (20000 Mb 全二重)	18000 (19555 ピーク) [1]	2162 (2331 ピーク) [1]
FDDI、MTU 4352 (デフォルト)	100	97	11.6
ATM 155、MTU 1500	155 (310 Mb 全二重)	180	21.5
ATM 155、MTU 9180 (デフォルト)	155 (310 Mb 全二重)	236	28.2
ATM 622、MTU 1500	622 (1244 Mb 全二重)	476	56.7
ATM 622、MTU 9180 (デフォルト)	622 (1244 Mb 全二重)	884	105

[1] 表に示す値は、専用パーティションで専用アダプターの場合の速度です。仮想イーサネット・アダプター (VIOS 内) または共用イーサネット・アダプター (SEA) の場合、または共用区画 (共用 LPAR) の場合の 10 ギガビット・イーサネット・アダプターのパフォーマンスは、この表には示しません。パフォーマンスは他の変数およびチューニングに影響されますが、それらはこの表の範囲外です。

注**:**

1. ピークの数値は、各方向に複数の TCP セッションが実行されている状態での最良のスループットを示します。その他の数値は、単一 TCP セッションに関する速度です。 単一セッションの速度は、プロセッサーの周波数、特定のアダプター、および使用する PCI スロットのタイプによって異なります。
2. 1000 M ビット・イーサネット (ギガビット・イーサネット) 二重通信速度は、PCI-eXtended (PCI-X) アダプターまたは PCI express (PCIe) アダプター・スロットに関する数値です。 PCI スロットの PCI アダプターや PCI-X アダプターの二重通信ワークロードの場合は、パフォーマンスが落ちます。10 Gb イーサネットの速度は、PCIe アダプターの場合のみが示されています。
3. データ転送速度は、インターネット・プロトコル・バージョン 4 (IPv4) を使用する TCP/IP の場合です。 **RFC1323** オプションは以下のアダプターの場合に使用可能です。
   - MTU サイズ 4096 以上のアダプター
   - 10 ギガビット・イーサネットまたはそれより高速なアダプター
4. ペイロード・レート (Mb) 欄の単位は、1 秒当たりのメガビット数です。ここで、1 メガビットは 1,000,000 ビットです。ペイロード・レート (MB) 欄の単位は、1 秒当たりのメガバイト数です。ここで、1 メガバイトは 1,048,576 バイトです。

アダプターとデバイスの設定

アダプターやデバイスの適切な運用と最高のパフォーマンスを実現するために重要なオプションがいくつかあります。

一般に AIX のデバイスには、ほとんどのインストール・システムに適したデフォルト値が用意されています。 したがって、これらのデバイス値を通常変更する必要はありません。 しかし、会社によっては、特有のネットワーク設定を必要とするポリシーがありますし、ネットワーク装置によっては、デフォルト値の変更を必要とする場合があります。

アダプター速度と二重モードの設定

AIX のデフォルト設定は Auto_Negotiation です。この場合は、最高のデータ転送速度を得るために速度と二重値の設定に関するネゴシエーションが行われます。 Auto_Negotiation モードを正しく機能させるためには、反対側の端点 (スイッチ) も Auto_Negotiation モードで構成する必要があります。

イーサネット・アダプターは、以下のモードで構成できます。

- 10_Half_Duplex
- 10_Full_Duplex
- 100_Half_Duplex
- 100_Full_Duplex
- Auto_Negotiation

ケーブルの両端、つまりアダプターとその反対側の端点 (通常はイーサネット・スイッチか、イーサネット・スイッチなしの 2 地点間構成の場合は別のアダプター) を同じ方法で構成することが重要です。 片方の端点の速度と二重モードを手動で設定した場合は、反対側の端点も、同じ速度と二重モードを手動で設定する必要があります。 片方の端点を手動で設定し、もう一方を Auto_Negotiation モードで設定すると、通常、リンクのパフォーマンスを落とすような問題が発生します。

可能な場合は、Auto_Negotiation モードを使用するのが最善です。これは、ほとんどのイーサネット・スイッチのデフォルトの設定です。 しかし、いくつかの 10/100 イーサネット・スイッチでは、二重モードの Auto_Negotiation モードがサポートされていません。 この種のスイッチの場合は、両方の端点で最適な速度と二重モードを手動で設定する必要があります。

注: 10 ギガビット・イーサネット・アダプターは、SR および LR ファイバー・メディアでは 1 つのスピードででしか作動できないので、Auto_Negotiation モードをサポートしていません。

イーサネット・スイッチのポート設定を表示し、ポート速度と二重モードの設定を変更するには、それぞれのイーサネット・スイッチに特有のコマンドを使用する必要があります。 これらのコマンドについては、スイッチ・ベンダーの資料を参照してください。

AIX では、アダプターの設定を変更するために、**smitty devices** コマンドを使用できます。 **netstat -v** コマンドまたは **entstat -d en***X* (*X* はイーサネット・インターフェース番号) コマンドを使用すれば、設定とネゴシエーション・モードを表示できます。 以下に、**entstat -d en3** コマンドの出力例の一部を示します。

```
10/100/1000 Base-TX PCI-X Adapter (14106902) Specific Statistics:
--------------------------------------------------------------------
Link Status: Up
Media Speed Selected: Auto negotiation
Media Speed Running: 1000 Mbps Full Duplex
```

アダプターの MTU 設定

VLAN タグを使用している物理または論理ネットワーク上のすべてのデバイスは、同じ MTU サイズを使用する必要があります。これは、ワイヤー上で送信できるフレーム (またはパケット) の最大サイズです。

MTU サイズのサポートは、ネットワーク・アダプターによって異なるので、ネットワーク上のすべてのデバイスで同じ MTU サイズを使用するための確認が必要になります。 例えば、ネットワーク上の他のアダプターがデフォルト MTU サイズ 1500 バイトを使用する場合、MTU サイズ 9000 バイトのジャンボ・フレームを使用するギガビット・イーサネット・アダプターを使用することはできません。 10/100 イーサネット・アダプターはジャンボ・フレーム・モードをサポートしないため、このギガビット・イーサネット・オプションとの互換性はありません。 さらに、イーサネット・スイッチでジャンボ・フレームがサポートされている場合は、イーサネット・スイッチでジャンボ・フレームを使用するための構成作業が必要です。

ネットワーク設定の早い段階でアダプターの MTU サイズを選択することは重要です。これによってすべてのデバイスとスイッチを適切に構成できるようになるからです。 さらに、多くの AIX チューニング・オプションは、選択した MTU サイズに依存します。

MTU サイズによるパフォーマンスへの影響

ネットワークの MTU サイズは、パフォーマンスに大きな影響を与えます。

大きな MTU サイズを使用すれば、オペレーティング・システムは、大きなサイズのパケットを少数送信することによって、同じネットワーク・スループットを達成できます。 ワークロードで大きなメッセージを送信できるのであれば、大きなパケットを使用することによって、オペレーティング・システムの処理を削減できます。 ワークロードが小さなメッセージしか送信できない場合は、大きな MTU サイズを設定しても意味はありません。

可能であれば、アダプターとネットワークがサポートする最大の MTU サイズを使用してください。例えば、非同期伝送モード (ATM) アダプターでは、デフォルトの MTU サイズ 9180 は、MTU サイズ 1500 バイト (一般に LAN エミュレーションで使用される) よりもはるかに効率的です。 ギガビットおよび 10 ギガビット・イーサネットの場合、ネットワーク上のすべてのマシンがギガビット・イーサネット・アダプターを持っていて、10/100 アダプターがネットワーク上に存在しないのであれば、ジャンボ・フレーム・モードの使用が最適です。 例えば、コンピューター研究室のサーバー間の接続には、ジャンボ・フレームを使用するのが一般的です。

ギガビット・イーサネット上のジャンボ・フレーム・モードの選択

ジャンボ・フレーム・モードはデバイス・オプションとして選択する必要があります。

ifconfig コマンドを使用して MTU サイズを変更しようとしても、うまくいきません。 SMIT を使用してアダプターの設定を表示する手順は、以下のとおりです。

1. 「**装置**」を選択します。
2. 「**通信**」を選択します。
3. 「**アダプター・タイプ**」を選択します。
4. 「**イーサネット・アダプターの特性の変更/表示**」を選択します。
5. 「**ジャンボ・フレームの送信**」オプションを「**いいえ**」から「**はい**」に変更します。

SMIT 画面の例は、次のとおりです。

```
                        Change/Show Characteristics of an Ethernet Adapter

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                [Entry Fields]
  Ethernet Adapter                               ent0
  Description                                    10/100/1000 Base-TX PCI-X Adapter (14106902)
  Status                                         Available
  Location                                       1H-08
  Receive descriptor queue size                 [1024]                                          +#
  Transmit descriptor queue size                [512]                                           +#
  Software transmit queue size                  [8192]                                          +#
```

```
  Transmit jumbo frames                          yes                                              +
  Enable hardware transmit TCP resegmentation    yes                                              +
  Enable hardware transmit and receive checksum  yes                                              +
  Media Speed                                    Auto_Negotiation                                 +
  Enable ALTERNATE ETHERNET address              no                                               +
  ALTERNATE ETHERNET address                    [0x000000000000]                                  +
  Apply change to DATABASE only                  no                                               +

F1=Help                  F2=Refresh               F3=Cancel                F4=List
Esc+5=Reset              Esc+6=Command            Esc+7=Edit               Esc+8=Image
Esc+9=Shell              Esc+0=Exit               Enter=Do
```

no コマンドを使用したネットワーク・パフォーマンス・チューニング

ネットワーク・オプション (**no**) コマンドによって、グローバル・ネットワーク・オプションの表示、変更、管理を実行できます。

以下の **no** コマンド・オプションを使用して、チューニング・パラメーターを変更できます。

オプション
　　定義

-a　　すべてのチューナブル・オプションとそれぞれの現在値を表示します。

-d [*tunable***]**
　　指定のチューナブル・オプションをデフォルト値に設定します。

-D　　すべてのオプションをデフォルト値に設定します。

-o *tunable***=[New Value]**
　　値を表示するか、指定のチューナブル・オプションを指定の新しい値に設定します。

-h [*tunable***]**
　　tunable を指定した場合は、そのチューナブル・パラメーターのヘルプを表示します。そうでない場合は、**no** コマンドの使用法を表示します。

-r　　**-o** オプションと共に使用して、リブート・タイプのチューナブル・オプションを nextboot ファイル内で永続化します。

-p　　**-o** オプションと共に使用して、動的チューナブル・オプションを nextboot ファイル内で永続化します。

-L *[tunable]*
　　-o オプションと共に使用して、1 つまたはすべてのチューナブル・オプションの特性を 1 行ごとにリストします。

以下に、**no** コマンドの例を示します。

```
NAME                      CUR    DEF    BOOT   MIN    MAX    UNIT           TYPE     DEPENDENCIES
--------------------------------------------------------------------------------------------------

General Network Parameters

--------------------------------------------------------------------------------------------------
sockthresh                85     85     85     0      100    %_of_thewall      D
--------------------------------------------------------------------------------------------------
fasttimo                  200    200    200    50     200    millisecond       D
--------------------------------------------------------------------------------------------------
inet_stack_size           16     16     16     1             kbyte             R
--------------------------------------------------------------------------------------------------
...lines omitted....
```

ここで、
CUR = current value

```
DEF = default value

BOOT = reboot value

MIN = minimal value

MAX = maximum value

UNIT = tunable unit of measure

TYPE = parameter type: D (for Dynamic), S (for Static), R for Reboot),B (for Bosboot), M (for Mount),
                       I (for Incremental) and C (for Connect)

DEPENDENCIES = list of dependent tunable parameters, one per line
```

ネットワーク属性のなかには、随時変更できるランタイム属性もあります。 ほかの属性はロード・タイム属性で、**netinet** カーネル・エクステンションをロードする前に設定する必要があります。

注**: no** コマンドを使用してパラメーターを変更した場合、動的パラメーターの変更はメモリー内で行われ、その変更が有効なのは次にシステムをブートするまでです。 ブートした時点で、すべてのパラメーターがリブート設定に設定されます。 動的パラメーターを永続化するために、**no** コマンドの **-r** または **-p** オプションを使用して、nextboot ファイルにオプションを設定します。リブート・パラメーター・オプションを有効にするには、システムのリブートが必要です。

no コマンドの詳細については、*Commands Reference, Volume 4*の『no コマンド』を参照してください。

TCP 高速パス・ループバック

伝送制御プロトコル (TCP) 高速パス・ループバック・オプションは、ループバック・トラフィックのパフォーマンスを改善するために使用されます。

tcp_fastlo ネットワーク・チューナブル・パラメーターは、TCP ループバック・トラフィックが、パフォーマンスを改善するために TCP/IP スタック全体 (プロトコルとインターフェース) の距離を削減することを可能にします。

このオプションを使用する場合、アプリケーションには変更は必要ありません。このオプションが使用可能になると、TCP ループバック・トラフィックは、UNIX ドメイン実装と同じように処理されます。

2 番目のオプション tcp_fastlo_crosswpar は、ワークロード・パーティション (wpar) 間で TCP 高速パス・ループバックが機能できるようにします。 tcp_fastlo_crosswpar オプションが機能するには、tcp_fastlo オプションが使用可能でなければなりません。

TCP ループバック・トラフィックの高速パスを使用可能にするには、**no** コマンドを使用して以下のように入力します。

```
# no -o tcp_fastlo=1
```

このオプションは動的であり、以降の TCP 接続に有効です。

ワークロード・パーティション (wpar) 間で TCP ループバック・トラフィックの高速パスを使用可能にするには、**no** コマンドを使用して以下のように入力します。

```
# no -o tcp_fastlo_crosswpar=1
```

注**:** この 2 つのオプション tcp_fastlo および tcp_fastlo_crosswpar は現在、デフォルトで使用不可です (0 に設定されています)。 これらのオプションは、将来の AIX リリースのために予約されています。

TCP 高速パス・ループバック・トラフィックは、TCP 接続がオープンである場合、**netstat** コマンドによる別個の統計で算出されます。 ループバック・インターフェースには算出されません。ただし、TCP 高速パス・ループバックは、TCP/IP およびループバック・デバイスを使用して、高速パス接続の確立と終了を行います。したがって、これらのパケットは通常どおりに算出されます。

割り込みの回避

割り込み処理は、ホストの CPU サイクルの点から見ると不経済です。

割り込みを処理するためには、システムはその前のマシン状態を保管し、割り込みがどこからのものかを判別し、さまざまなハウスキーピング・タスクを実行し、適切なデバイス・ドライバー割り込みハンドラーを呼び出さなければなりません。 デバイス・ドライバーは通常、アダプターの割り込み状況レジスターの読み取り (マシン速度と比較して遅い)、SMP のロックの取得、バッファーの取得と解放など、オーバーヘッドの大きい操作を実行します。

AIX のほとんどのデバイス・ドライバーは、送信完了割り込みを使用しません。こうすることで送信パケットでの割り込みを回避しています。 送信完了処理は通常は次の送信操作で行われるので、個々の送信完了割り込みが回避されます。 **netstat -v**、**entstat**、**atmstat**、または **fddistat** のようなコマンドを使用して、送信および受信パケット数と、送信および受信割り込み数について、それぞれの状況を表示することができます。この統計情報から、送信割り込みが回避されていることが明確に分かります。 サード・パーティーのアダプターおよびドライバーによっては、この規則に従っていないものもあります。

LAN アダプター上のドッグ・スレッドの使用可能化

ドッグ・スレッド 機能を使用可能にすると、ドライバーは着信パケットをスレッドのキューに入れ、スレッドは IP、TCP、およびソケット・コードの呼び出しを処理します。

デフォルトでは、ドライバーが IP を直接呼び出し、割り込みレベルでの実行中に、IP がプロトコル・スタックをソケット・レベルに呼び出します。 これにより、命令パス長が最短になりますが、割り込み保持時間は長くなります。 SMP システムでは、単一の CPU が、高速アダプターからパケットを受け取るときのボトルネックになる可能性があります。 スレッドは、アイドル状態になっている可能性のあるほかの CPU で実行できます。 着信パケットの速度が速いシステムでドッグ・スレッドを使用可能にすると、システムの能力が上がって、着信パケットを複数の CPU で並行処理できることがあります。

ドッグ・スレッド機能のマイナス面として、パケットがスレッドのキューに入れられ、そのスレッドがディスパッチされることになるので、軽い負荷の場合の待ち時間が長くなり、ホスト CPU の使用率も上がることが挙げられます。

注**:** このフィーチャーは、パス長だけを長くし、パフォーマンスを低下させるので、ユニプロセッサーではサポートされません。

これは、LAN アダプターの入力側 (受信) のフィーチャーです。 **ifconfig** コマンド (**ifconfig** *interface* **thread** または **ifconfig** *interface hostname* **up thread**) を使用すると、インターフェース・レベルで構成できます。

このフィーチャーを使用不可にするには、**ifconfig** *interface* **-thread** コマンドを使用します。以下に例を示します。

```
# ifconfig en0 thread

# ifconfig en0
en0: flags=5e080863,e0<UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST,GROUPRT,64BIT,CHECKSUM_OFFLOAD,PSEG,THREAD,CHAIN>
        inet 192.1.0.1 netmask 0xffffff00 broadcast 192.1.0.255

# ifconfig en0 -thread
```

```
# ifconfig en0
en0: flags=5e080863,c0<UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST,GROUPRT,64BIT,CHECKSUM_OFFLOAD,PSEG,THREAD,CHAIN>
        inet 192.1.0.1 netmask 0xffffff00 broadcast 192.1.0.255
```

netstat -s コマンドは、スレッドによって処理されたパケットの数、およびスレッドのキューが着信パケットを破棄したかどうかを示すカウンター値も表示します。 以下に、**netstat -s** コマンドの例を示します。

```
# netstat -s| grep hread

        352 packets processed by threads
        0 packets dropped by threads
```

ドッグ・スレッドを使用する場合のガイドラインは次のとおりです。

- アダプターより多くの CPU をインストールする必要があります。 一般的には、アダプターより 2 倍以上多い CPU が推奨されます。
- システムの CPU が高速であるほど、利点は少なくなります。 マシンの CPU 速度が低速であるほど、多くの利点を得られます。
- このフィーチャーを使用すると、ほとんどの場合、入力パケットの速度が速いときにパフォーマンスが向上します。 パケットの速度は、MTU ネットワークが小さいと高速になるので、MTU が 9000 ギガビット (ジャンボ・フレーム) の場合よりも MTU が 1500 ギガビットの方がパフォーマンスが向上します。

  ドッグ・スレッドは、キュー上での作業が多く、かつ (入力を待機して) スリープ状態になる必要がないときに、実行状態が最もよくなります。 これにより、スレッドをウェイクアップするドライバーのオーバーヘッド、およびスレッドをディスパッチするシステムのオーバーヘッドがなくなります。
- ドッグ・スレッドによって、特定の CPU が割り込みにマスクをして送信する時間が短縮されることもあります。 これにより、CPU が解放され、通常のユーザー・レベル作業の再開が早くなります。
- パケットの速度がスレッドを実行させ続けるのに十分な速さでない場合は、ドッグ・スレッドによってパフォーマンスが約 10% 低下することがあります。 10% は、スレッドのスケジュールを作成し、スレッドをディスパッチするために必要な CPU オーバーヘッドが増えたときの平均値です。

インターフェース固有のネットワーク・オプション

インターフェース固有のネットワーク・オプション (ISNO) は、最高のパフォーマンスを得るために、IP ネットワーク・インターフェースをカスタマイズできます。

個々のインターフェースに設定される値は、**no** コマンドで設定されるシステム全体にわたる値よりも優先されます。 このフィーチャーは、**no** コマンドの **use_isno** オプションを使用して、システム全体で使用可能にしたり (デフォルト)、使用不可にしたりすることができます。 この単一ポイントの ISNO 使用不可オプションは、システム管理者がパフォーマンス上の問題を特定する必要があるときに、潜在的なチューニング・エラーを防ぐための診断ツールとして組み込まれています。

プログラマーとパフォーマンス分析者は、TCP 接続が確立されるまで、ISNO 値がソケットに現れない (つまり **getsockopt()** システム・コールで読み取ることができない) ことに注意してください。 このソケットが実際に使用する特定のネットワーク・インターフェースは、接続が完了するまで認識されないので、ソケットは、**no** コマンドから取得するシステムのデフォルトを反映します。 TCP 接続が受け付けられ、ネットワーク・インターフェースが認識された後、ISNO 値がソケットに書き込まれます。

以下のパラメーターは、サポートされているネットワーク・インターフェースごとに追加されており、TCP (UDP でない) 接続でのみ有効です。

- **rfc1323**

- **tcp_nodelay**
- **tcp_sendspace**
- **tcp_recvspace**
- **tcp_mssdflt**

特定のインターフェース用に設定すると、これらの値によって、システムに設定された対応する **no** オプション値が変更されます。 これらのパラメーターは、すべてのメイン・ストリーム TCP/IP インターフェース (トークンリング、FDDI、10/100 イーサネット、およびギガビット・イーサネット) に使用できます。ただし、SP スイッチ上の **css#** IP は除きます。 簡単な回避策として、SP スイッチのユーザーは、システム全体に影響する **no** コマンドを実行することによって、スイッチに適したチューニング・オプションを設定できます。次に、ISNO を使用して、その他のシステム・インターフェースに必要な値を設定できます。

これらのオプションは TCP/IP インターフェース (en0 または tr0 など) 用に設定されており、ネットワーク・アダプター (ent0 または tok0 など) 用には設定されていません。

AIX は、MTU 1500 およびジャンボ・フレーム・モード (MTU 9000) のギガビット・イーサネット・インターフェースのデフォルト値を設定します。 **SMIT tcpip** 画面でインターフェースを構成した場合は、ISNO オプションがデフォルト値に設定されているはずです (その場合は、優れたパフォーマンスを発揮します)。

10/100 イーサネット・アダプターやトークンリング・アダプターの場合は、基本的にシステム・グローバルの **no** のデフォルトで正しく動作するため、システムは ISNO デフォルトを設定しません。 しかし、必要があれば、ISNO 属性を設定して、グローバル・デフォルトをオーバーライドすることも可能です。

以下のサンプルは、MTU 1500 モードのギガビット・イーサネットの **tcp_sendspace** および **tcp_recvspace** のデフォルト ISNO 値を表示します。

```
# ifconfig en0
en0: flags=5e080863,c0<UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST,GROUPRT,64BIT,CHECKSUM_OFFLOAD,PSEG,CHAIN>
        inet 10.0.0.1 netmask 0xffffff00 broadcast 192.0.0.255
        tcp_sendspace 131072 tcp_recvspace 65536
```

ジャンボ・フレーム・モードでは、**tcp_sendspace**、**tcp_recvspace**、および **rfc1323** のデフォルト ISNO 値は以下のように設定されます。

```
 # ifconfig en0
en0: flags=5e080863,c0<UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST,GROUPRT,64BIT,CHECKSUM_OFFLOAD,PSEG,CHAIN>
       inet 192.0.0.1 netmask 0xffffff00 broadcast 192.0.0.255
       tcp_sendspace 262144 tcp_recvspace 131072 rfc1323 1
```

MTU サイズが 4096 バイト以上の場合に **rfc1323** を使用可能にし、高速のアダプター (ギガビット以上) に対して **tcp_sendspace** および **tcp_recvspace** の値を少なくとも 128 KB に設定するには、次の設定値を使用します。 非常に高速のアダプターは 256 KB に設定されます。 「ブランク」の値は、オプションが設定されないためグローバルの「no」の設定を継承することを意味します。

インターフェース	速度	**MTU**	**tcp_sendspace**	**tcp_recvspace**	**rfc1323**	**tcp_nodelay**	**tcp_mssdflt**
lo0 (ループバック)	N/A	16896	131072	131072	1		
イーサネット	10 または 100 (M ビット)						
イーサネット	1000 (ギガビット)	1500	131072	65536	1		
イーサネット	1000 (ギガビット)	9000	262144	131072	1		
イーサネット	10 GigE	1500	262144	262144	1		
イーサネット	10 GigE	9000	262144	262144	1		
イーサチャネル	基盤のインターフェースの速度/MTU を基に構成されます。						
仮想イーサネット	N/A	任意	262144	262144	1		
InfiniBand	N/A	2044	131072	131072	1		

以下の方法で ISNO オプションを設定できます。

- SMIT
- **chdev** コマンド
- **ifconfig** コマンド

SMIT または **chdev** コマンドを使用すれば、ディスク上の ODM データベース内の値を変更できるので、これらの値は永続化されます。 **ifconfig** コマンドは、メモリー上の値だけを変更するので、これらの値は次のリブート時に、 ODM に格納されている以前の値に戻されます。

SMIT による **ISNO** のオプションの変更**:**

SMIT を使用して、ISNO オプションを変更できます。

コマンド・ラインで以下のように入力します。

```
# smitty tcpip
```

1. 「**追加構成**」オプションを選択します。
2. 「**ネットワーク・インターフェース**」オプションを選択します。
3. 「**ネットワーク・インターフェースの選択**」を選択します。
4. 「**ネットワーク・インターフェースの特性の変更**」を選択します。
5. カーソルを使ってインターフェースを選択します。 例えば、「en0」を選択します。

その後、以下の画面が表示されます。

```
Change / Show a Standard Ethernet Interface

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                [Entry Fields]
  Network Interface Name                         en0
  INTERNET ADDRESS (dotted decimal)             [192.0.0.1]
  Network MASK (hexadecimal or dotted decimal)  [255.255.255.0]
  Current STATE                                  up                          +
  Use Address Resolution Protocol (ARP)?         yes                         +
  BROADCAST ADDRESS (dotted decimal)            []
  Interface Specific Network Options
```

```
    ('NULL' will unset the option)
    rfc1323                                          []
    tcp_mssdflt                                      []
    tcp_nodelay                                      []
    tcp_recvspace                                    []
    tcp_sendspace                                    []

F1=Help                     F2=Refresh                  F3=Cancel                   F4=List
Esc+5=Reset                 Esc+6=Command               Esc+7=Edit                  Esc+8=Image
Esc+9=Shell                 Esc+0=Exit                  Enter=Do
```

ISNO システム・デフォルトは内部的には設定されますが、表示されないことに注意してください。 この例では、**tcp_sendspace** のデフォルト値をオーバーライドして、65536 という値に下げます。

smitty tcpip を使用してインターフェースに戻り、「最小構成と始動」を選択します。 「en0」を選択し、インターフェースが最初に設定された時点のデフォルト値を取得します。

ifconfig コマンドを使用して ISNO オプションを表示すれば、**tcp_sendspace** 属性に 65536 が設定されていることを確認できます。 次に例を示します。

```
# ifconfig en0
en0: flags=5e080863,c0<UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST,GROUPRT,64BIT,CHECKSUM_OFFLOAD,PSEG,CHAIN>
         inet 192.0.0.1 netmask 0xffffff00 broadcast 192.0.0.255
         tcp_sendspace 65536 tcp_recvspace 65536
```

lsattr コマンドの出力からも、この属性のシステム・デフォルトがオーバーライドされていることを確認できます。

```
# lsattr -E -l en0
alias4                      IPv4 Alias including Subnet Mask           True
alias6                      IPv6 Alias including Prefix Length         True
arp           on            Address Resolution Protocol (ARP)          True
authority                   Authorized Users                           True
broadcast                   Broadcast Address                          True
mtu           1500          Maximum IP Packet Size for This Device     True
netaddr       192.0.0.1     Internet Address                           True
netaddr6                    IPv6 Internet Address                      True
netmask       255.255.255.0 Subnet Mask                                True
prefixlen                   Prefix Length for IPv6 Internet Address    True
remmtu        576           Maximum IP Packet Size for REMOTE Networks True
rfc1323                     Enable/Disable TCP RFC 1323 Window Scaling True
security      none          Security Level                             True
state         up            Current Interface Status                   True
tcp_mssdflt                 Set TCP Maximum Segment Size               True
tcp_nodelay                 Enable/Disable TCP_NODELAY Option          True
tcp_recvspace               Set Socket Buffer Space for Receiving      True
tcp_sendspace 65536         Set Socket Buffer Space for Sending        True
```

chdev および **ifconfig** コマンドによる **ISNO** オプションの変更**:**

以下のコマンドを使用して、最初にシステムとインターフェースのサポートを検証してから、新しい値の設定と検証を実行できます。

- 次のコマンドを使用して、**use_isno** オプションが使用可能になっていることを確認します。

  ```
  # no -a | grep isno
  use_isno = 1
  ```

- 次のように **lsattr -El** コマンドを使用して、インターフェースが 5 つの新しい ISNO をサポートするようにします。

  ```
  # lsattr -E -l en0 -H
  attribute           value description                          user_settable
  :
  rfc1323             Enable/Disable TCP RFC 1323 Window Scaling True
  ```

```
tcp_mssdflt         Set TCP Maximum Segment Size              True
tcp_nodelay         Enable/Disable TCP_NODELAY Option         True
tcp_recvspace       Set Socket Buffer Space for Receiving     True
tcp_sendspace       Set Socket Buffer Space for Sending       True
```

- **ifconfig** または **chdev** コマンドを使用して、インターフェース固有の値を設定します。 **ifconfig** コマンドは、一時的に値を設定します (テストに最適です)。 **chdev** コマンドは、ODM を変更するので、システムのリブート後にカスタム値に戻ります。

  例えば、**tcp_recvspace** および **tcp_sendspace** を 64 KB に設定し、**tcp_nodelay** を使用可能にするには、次のいずれかの方法を使用します。

  ```
  # ifconfig en0 tcp_recvspace 65536 tcp_sendspace 65536 tcp_nodelay 1
  ```

  または

  ```
  # chdev -l en0 -a tcp_recvspace=65536 -a tcp_sendspace=65536 -a tcp_nodelay=1
  ```

- 次のように **ifconfig** または **lsattr** コマンドを使用して、設定を検証します。

  ```
  # ifconfig en0
  en0: flags=5e080863,c0<UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST,GROUPRT,64BIT,CHECKSUM_OFFLOAD,PSEG,CHAIN>
          inet 9.19.161.100 netmask 0xffffff00 broadcast 9.19.161.255
          tcp_sendspace 65536 tcp_recvspace 65536 tcp_nodelay 1
  ```

  または

  ```
  # lsattr -El en0
  rfc1323             Enable/Disable TCP RFC 1323 Window Scaling True
  tcp_mssdflt         Set TCP Maximum Segment Size              True
  tcp_nodelay   1     Enable/Disable TCP_NODELAY Option         True
  tcp_recvspace 65536 Set Socket Buffer Space for Receiving     True
  tcp_sendspace 65536 Set Socket Buffer Space for Sending       True
  ```

TCP ワークロードのチューニング

TCP パフォーマンスに影響を与える AIX チューニング・オプション値がいくつかあります。

ftp や **rcp** コマンドなど、信頼できるトランスポート・コントロール・プロトコル (TCP) を使用するアプリケーションは少なくありません。

注**: no -o** コマンドは、 TCP/IP 接続に影響するチューニング・オプションを変更した場合に、これらの変更が、変更後に確立された接続のみに有効であることを通知します。 また、**inetd** デーモンが listen する新しい接続のためのプロセスに影響を与えるオプションが変更された場合、**no -o** コマンドは、**inetd** デーモン・プロセスを再始動します。

TCP ストリーミング・ワークロードのチューニング**:**

ストリーミング・ワークロードは、1 つの端点から他の端点へ大量のデータを移動します。 ストリーミング・ワークロードの例は、ファイル転送、バックアップまたはリストアのワークロード、バルク・データ転送などです。これらのワークロードで重要な意味を持つ測定基準は帯域幅ですが、終端間の待ち時間も注目に値します。

ストリーミング・アプリケーションの TCP パフォーマンスに影響を与える主なチューニング・オプションは、以下のとおりです。

- **tcp_recvspace**
- **tcp_sendspace**
- **rfc1323**
- **MTU path discovery**

- **tcp_nodelayack**
- **sb_max**
- チェックサム・オフロードや TCP 大容量送信などのアダプター・オプション

アダプターのタイプと MTU サイズに基づいて最適なパフォーマンスを得るためのチューニング・オプション値の推奨サイズを以下の表にまとめます。

Device	速度	**MTU** サイズ	**tcp_sendspace**	**tcp_recvspace**	**sb_max** [1]	**rfc1323**
トークンリング	4 または 16 M ビット	1492	16384	16384	32768	0
イーサネット	10 M ビット	1500	16384	16384	32768	0
イーサネット	100 M ビット	1500	16384	16384	65536	0
イーサネット	ギガビット	1500	131072	65536	131072	0
イーサネット	ギガビット	9000	131072	65535	262144	0
イーサネット	ギガビット	9000	262144	131072[2]	524288	1
イーサネット	10 ギガビット	1500	131072	65536	131072	0
イーサネット	10 ギガビット	9000	262144	131072	262144	1
ATM	155 M ビット	1500	16384	16384	131072	0
ATM	155 M ビット	9180	65535	65535[3]	131072	0
ATM	155 M ビット	65527	655360	655360[4]	1310720	1
FDDI	100 M ビット	4352	45056	45056	90012	0
ファイバー・チャネル	2 ギガビット	65280	655360	655360	1310720	1

(1) *sb_max* チューニング・オプションにはデフォルト値 1048576 を使用することを推奨します。 この表の値は、*sb_max* チューニング・オプションで受け入れ可能な最小値です。

(2) ギガビット・イーサネット上のジャンボ・フレームで、rfc1323 を使用可能にしてこれらのオプションを使用すると、パフォーマンスが若干向上します。

(3) TCP 送信スペースと受信スペースを特定の組み合わせにすると、結果的にスループットが非常に低下します (1 M ビット以下)。 この問題を回避するには、*tcp_sendspace* チューニング・オプションを、MTU サイズの 3 倍以上の値、または受信側の *tcp_recvspace* と同じ値に設定します。

(4) TCP はウィンドウ・サイズとして 16 ビット値しか使用しません。 これは、最大ウィンドウ・サイズの 65536 バイトに変換されます。 MTU サイズが大きい (例えば、32 KB または 64 KB などの) アダプターの場合は、TCP ストリーミング・パフォーマンスが非常に低下することがあります。 例えば、MTU サイズが 64 KB のデバイスでは、*tcp_recvspace* が 64 KB に設定されていると、TCP は 1 つのパケットだけを送信でき、そのウィンドウはクローズします。 再び送信するには、受信側から ACK が戻されるのを待機する必要があります。 この問題は、次の方法のうちのいずれかで解消できます。

- *rfc1323* を使用可能にする。この場合は TCP が拡張され、64 KB を超えるサイズのウィンドウを使用できるように 16 ビットの制限を超えることができます。そのあと、*tcp_recvspace* チューニング・オプションを MTU サイズの 10 倍などの大きい値に設定すると、TCP がデータのストリーミングを行い、高いパフォーマンスが得られます。
- アダプターの MTU サイズを小さくする。 例えば、**ifconfig at0 mtu 16384** コマンドを使用して、ATM MTU サイズを 16 KB に設定してください。 これにより、TCP はより小さいサイズの MSS 値を算出できます。 MTU サイズが 16 KB でも、 TCP は 64 KB ウィンドウ・サイズに 4 個のパケットを送信できます。

以下に TCP ストリーミング・ワークロードのチューニングに関する一般的なガイドラインを示します。

- TCP 送信および受信スペースを少なくとも MTU サイズの 10 倍に設定します。
- MTU サイズが 8 KB を超えている場合は、TCP 受信スペース値を大きくするために、*rfc1323* を使用可能にする必要があります。
- 高速アダプターでは、TCP 送信および受信スペース値を大きくすることでパフォーマンスを改善できます。
- 高速アダプターでは、*tcp_sendspace* チューニング・オプションの値は、 *tcp_recvspace* 値の 2 倍である必要があります。
- lo0 インターフェースの *rfc1323* はデフォルトで設定されます。 **lo0** のデフォルト MTU サイズは 1500 より高くなるため、*tcp_sendspace* および *tcp_recvspace* チューナブルは 128K に設定されます。

ftp および **rcp** コマンドは、*tcp_sendspace* および *tcp_recvspace* チューニング・オプションをチューニングすることによって優れた効果を得られる TCP アプリケーションの例です。

tcp_recvspace チューニング・オプション*:*

tcp_recvspace チューニング・オプションは、カーネル内で受信システムが受信ソケット・キューにバッファリングできるデータのバイト数を指定します。

tcp_recvspace チューニング・オプションは TCP プロトコルが TCP ウィンドウ・サイズを設定する際にも使用されます。TCP はこの TCP ウィンドウ・サイズを使用して、受信側に送信するデータのバイト数を制限し、受信側がデータをバッファーに入れるための十分なスペースを確保できるようにします。
tcp_recvspace チューニング・オプションは TCP パフォーマンスにとって重要なパラメーターです。TCP は複数のパケットをネットワークに送信して、ネットワーク・パイプラインが満杯になるようにする必要があるからです。 TCP がパイプラインに十分の量のパケットを保てないと、パフォーマンスに影響が出ます。

以下の方法で、*tcp_recvspace* チューニング・オプションを設定できます。

- プログラムからの **setsockopt()** システム・コール
- **no -o tcp_recvspace=[*value*]** コマンド
- *tcp_recvspace* ISNO パラメーター

tcp_recvspace チューニング・オプションを設定するときには、少なくとも MTU サイズの 1/10 の値に設定するというのが一般的な指針です。 *tcp_recvspace* チューニング・オプションを算出するには、帯域幅遅延の積値を 8 で除算します。次の式の計算を行います。

```
帯域幅遅延の積 = 容量 (ビット)= 帯域幅 (ビット/秒) x 往復時間 (秒)
```

容量値を 8 で除算した値は、ネットワーク・パイプラインを最大限に活用するために必要な TCP ウィンドウ・サイズの見積もりに適しています。 往復遅延が長くなりネットワークの速度は速くなればなるほど、帯域幅遅延の積値は大きくなり、したがって TCP ウィンドウも大きくなります。 その一例として、100 M ビットのネットワークで往復時間が 0.2 ミリ秒の場合を考えてみましょう。帯域幅遅延の積値は、次の式で計算できます。

```
帯域幅遅延の積 = 100000000 x 0.0002 = 20000
20000/8 = 2500
```

このように、上の例の場合では、TCP ウィンドウ・サイズは少なくとも 2500 バイトにする必要があります。 単一の LAN に 100 M ビットやギガビット・イーサネットが使用されている環境では、*tcp_recvspace* と *tcp_sendspace* チューニング・オプションの値を帯域幅遅延の積の計算値の少なくとも 2 から 3 倍にすると、最適なパフォーマンスが得られます。

tcp_sendspace チューニング・オプション*:*

tcp_sendspace チューニング・オプションは、送信側アプリケーションが 2 番目のコールをブロックする前に、カーネルのバッファーに入れることのできるデータの量を指定します。

TCP ソケット送信バッファーは、アプリケーション・データが TCP プロトコルによって受信側に送信される前に、mbuf/クラスターを使用してカーネルのバッファーにアプリケーション・データを入れるために使用されます。送信バッファーのデフォルトのサイズはパラメーター *tcp_sendspace* チューニング値によって指定されますが、プログラムが **setsockopt()** サブルーチンを使用してそれを指定変更することもできます。

tcp_sendspace チューニング値は少なくとも *tcp_recvspace* 値と同じ大きさに設定し、高速アダプターの場合には、*tcp_sendspace* 値を *tcp_recvspace* 値の少なくとも 2 倍のサイズにします。

アプリケーションがソケットに O_NDELAY または O_NONBLOCK を指定すると、入出力がブロックされなくなり、その後送信バッファーが満杯になると、アプリケーションはスリープ状態になることはなく、EWOULDBLOCK/EAGAIN エラーを伴って戻ります。このエラーを処理できるようにアプリケーションをコーディングする必要があります (推奨されるソリューションは、再送信を試行する間、短時間だけスリープ状態にすることです)。

rfc1323 チューニング・オプション*:*

rfc1323 チューニング・オプションは TCP ウィンドウ・スケーリング・オプションを使用可能にします。

TCP ウィンドウ・スケーリング・オプションは TCP ネゴシエーション・オプションの一種なので、TCP 接続の両方のエンドポイントでこのオプションを使用可能にすることにより有効になります。 デフォルトでは、TCP ウィンドウ・サイズは 65536 バイト (64K バイト) に制限されていますが、*rfc1323* 値が 1 に設定されている場合はそれより大きい値に設定することができます。*tcp_recvspace* 値を 65536 より大きい値に設定する場合は、接続の両側で *rfc1323* 値を 1 に設定してください。接続の両側で *rfc1323* 値を設定しないと、*tcp_recvspace* チューニング・オプションの有効な値は 65536 になります。 このオプションを使用した場合、TCP プロトコル・ヘッダーには 12 バイトが追加され、この分はユーザー・ペイロード・データから差し引かれます。そのため、MTU アダプターが小さい場合は、このオプションのためにパフォーマンスが若干低下する場合があります。

MTU サイズが大きい (例えば、32 K または 64 K など) アダプターを介してデータを送信する場合は、1 つのパケットで TCP ウィンドウ・サイズ全体を使ってしまうため、このオプションを使用可能にしないと、TCP ストリーミング・パフォーマンスが最適にならないことがあります。 そのため、パケットごとに受信側から TCP 肯定応答およびウィンドウの更新を待つ必要が生じるので、TCP は複数パケットをストリーミングできません。 コマンド **no -o rfc1323=1** を使用して *rfc1323* オプションを使用可能にすると、TCP のウィンドウ・サイズを 4 GB ほどに設定できます。 *rfc1323* オプションを 1 に設定したら、*tcp_recvspace* パラメーターを、MTU の 10 倍などのかなり大きい値に設定できます。

送信側と受信側のシステムが *rfc1323* オプションをサポートしない場合に、MTU サイズの大きいアダプターのストリーミング・パフォーマンスを向上させる 1 つの方法は、MTU サイズを減らすことです。 MTU サイズ 65536 を使用すると TCP が 1 つの未解決パケットに制限されてしまうので、その代わりに MTU サイズ 16384 を選択します。こうすると、*tcp_recvspace* 値が 65536 バイトの場合に、TCP は 4 つの未解決パケットを使うことができるので、パフォーマンスが向上します。 ただし、ネットワーク上のすべてのノードで同じ MTU サイズを使用する必要があります。

***TCP* パス *MTU* ディスカバリー:**

AIX では、TCP パス MTU ディスカバリー・プロトコル・オプションがデフォルトで使用可能です。 このオプションにより、プロトコル・スタックは、2 つのホスト間のパスに存在するネットワーク上の最小 MTU サイズを決定できます。このオプションは **tcp_pmtu_discover=1** ネットワーク・オプションによって制御されます。

TCP パス MTU ディスカバリーの実装には、ICMP ECHO メッセージではなく、接続そのものの TCP パケットが使用されます。TCP/IP カーネル・エクステンションは、PMTU ディスカバリー関連情報を保管するための、PMTU テーブルというテーブルを維持します。宛先への TCP 接続が確立されると、PMTU テーブルにその宛先のエントリーが作成されます。 PMTU 値は、発信インターフェースの MTU 値です。

TCP パケットは、IP ヘッダーの「Don't Fragment」(DF) ビットを設定して送信されます。 MTU 値が TCP パケットのサイズより小さいネットワーク・ルーターに TCP パケットが到達した場合、そのルーターは、フラグメント化できないためにメッセージを転送できないことを示す ICMP エラー・メッセージを送り返します。 エラー・メッセージを送信するルーターがコンパイルに RFC 1191 を使用していれば、ICMP エラー・メッセージにネットワークの MTU 値が含められます。 そうでない場合に TCP パケットを再送するには、AIX TCP/IP カーネル・エクステンション内の既知の MTU 値のテーブルから、もっと小さい値を MTU サイズに割り当てる必要があります。 それから PMTU テーブル内の宛先の PMTU 値が、小さくした新しい MTU サイズに更新され、TCP パケットが再送されます。以降のその宛先への TCP 接続では、更新された PMTU 値が使用されます。

pmtu コマンドを使用して、PMTU エントリーの表示または削除を行えます。 以下に、**pmtu** コマンドの例を示します。

```
# pmtu display

   dst           gw            If    pmtu     refcnt   redisc_t    exp

 --------------------------------------------------------------------------


10.10.1.3    10.10.1.5        en1    1500        2          9       0

10.10.2.5    10.10.2.33       en0    1500        1          0       0
```

使われていない PMTU エントリー (値が 0 の `refcnt` エントリー) は、PMTU テーブルが大きくなるのを防ぐために削除されます。 使われていないエントリーは、`refcnt` 値が 0 になってから *pmtu_expire* 分後に削除されます。*pmtu_expire* ネットワーク・オプションのデフォルト値は 10 分です。 PMTU エントリーが期限切れにならないようにするには、*pmtu_expire* 値を 0 に設定します。

この TCP パス MTU ディスカバリーのインプリメンテーションでは、経路のクローン作成は不要です。このことは、経路指定テーブルが小さくなり管理しやすくなることを意味します。

***tcp_nodelayack* チューニング・オプション:**

tcp_nodelayack オプションを設定すると、TCP は通常の 200 ミリ秒の遅延を行わず、即時に肯定応答を送信します。肯定応答を即時に送信すると多少オーバーヘッドが追加されることがありますが、場合によってはパフォーマンスが大幅に改善します。

TCP が肯定応答の送信を 200 ミリ秒遅延させると、送信側が受信側からの肯定応答を待つ一方で、受信側も送信側からさらにデータが送信されるのを待ち、こうしてパフォーマンス上の問題が生じます。 結果として、ストリーミングのスループットは低くなります。 この問題が起きていると思われる場合は、

tcp_nodelayack オプションを有効にして、ストリーミングのパフォーマンスが改善されるかどうかを確認してください。 パフォーマンスが改善されない場合は、*tcp_nodelayack* オプションを無効にします。

sb_max チューニング・オプション*:*

sb_max チューニング・オプションは個々のソケットのキューに入れられるソケット・バッファーの上限数を設定し、これにより送信側のソケットまたは受信側のソケットのキューに入れられるバッファーが使用するバッファー・スペース量を制御します。

システムは、バッファーの内容ではなく、バッファーのサイズに基づいて、使用するソケット・バッファーを算出します。

デバイス・ドライバーが 100 バイトのデータを 2048 バイトのバッファーに入れる場合は、システムで、2048 バイトのソケット・バッファー・スペースが使用されると見なされます。 一般的に、デバイス・ドライバーは、アダプター最大サイズ・パケットを受け取るのに十分な大きさのバッファーへ、バッファーを受信します。 これは多くの場合、バッファー・スペースの浪費になりますが、より小さいバッファーにデータをコピーするにはより多くの CPU サイクルが必要になります。

注**:** AIX では、*sb_max* チューニング・オプションのデフォルト値は 1048576 という大きな値です。 このパラメーターを変更する場合は、『TCP ストリーミング・ワークロードのチューニング』を参照して、*sb_max* の推奨値を確認してください。

TCP チェックサム・オフロード*:*

TCP チェックサム・オフロード・オプションを使用すると、ネットワーク・アダプターが送信および受信時の TCP チェックサムを計算できるようになり、AIX ホスト CPU はチェックサムを計算する必要がなくなります。

どれだけの節約になるかはパケット・サイズによって異なります。 パケットが小さい場合はこのオプションを使用してもほとんどまたはまったく節約にはなりませんが、パケットが大きい場合はかなりの節約になります。 PCI-X GigE アダプターでは、MTU 1500 の場合は CPU 使用率が通常約 5% 減り、MTU 9000 (ジャンボ・フレーム) の場合は CPU 使用率が約 15% の節約になります。

400 MHz より高速のプロセッサーを搭載したマシンで、TCP チェックサム・オフロード・オプションを有効にすると、MTU 1500 での TCP ストリーミング・スループットは遅くなります。それは、ホスト・システムが実行するチェックサムの速度がギガビット・イーサネット PCI アダプター FC2969 および FC2975 よりも速いためです。 そのため、これらのアダプターでは、このオプションはデフォルトでオフになっています。 これらのアダプターがジャンボ・フレームを使用する場合は、チェックサムの計算が必要であっても、ワイヤー・スピードで実行できます。

PCI-X ギガビット・イーサネット・アダプターは TCP チェックサム・オフロード・オプションを有効にするとワイヤー・スピードで実行でき、ホスト CPU の処理も軽減されるので、このオプションはデフォルトで有効になっています。

TCP 大容量送信オフロード*:*

TCP 大容量送信オフロード・オプションを使用すると、AIX TCP 層は最大 64 KB 長の TCP メッセージを作成できます。アダプターは IP およびイーサネット・デバイス・ドライバーを介して、そのメッセージを 1 コール分のスタックで送信できます。

次にアダプターは、ケーブルでデータを送信するためにメッセージを複数の TCP フレームに分割します。ケーブルで送信される TCP パケットは、1500 バイト・フレーム (MTU 1500 の場合) または最大 9000 バイト・フレーム (MTU 9000 (ジャンボ・フレーム) の場合) のいずれかです。

TCP 大容量送信オフロード・オプションを使用しない場合、1500 バイトのパケットを使用して 64 KB のデータを送信するには 44 コール分のスタックが必要です。TCP 大容量送信オプションを使用すると、最大 64K バイトのデータを 1 コール分のスタックで送信できるので、ホストの処理が軽減され、ホスト・プロセッサーのプロセッサー使用率が低くなります。次に、イーサネット・アダプターは TCP セグメンテーション・オフロードを実行して、データを MTU サイズのパケット (通常 1500 バイト) に分割します。どのくらいの節約になるかは、平均 TCP 大容量送信サイズによって異なります。 例えば、PCI-eXtended (PCI-X) ギガビット・イーサネット・アダプターを使用し、MTU サイズを 1500 にすると、ホスト・プロセッサー CPU が 60 から 75% 軽減されます。ジャンボ・フレーム (MTU 9000) の場合は、システムが既に大容量フレームを送信しているので、節約はそれほど多くありません。ジャンボ・フレームの場合の代表的な例では、ホスト・プロセッサー CPU の軽減は 40% です。

大容量送信オフロード・オプションをサポートするイーサネット・アダプターでは、占有モードで作業する場合にデフォルトでこのオプションが使用可能になります。データ・ストリームを管理するワークロード (ファイル転送プロトコル (FTP)、RCP、テープ・バックアップ、および類似の大量データ移動アプリケーション) を扱う場合、10 ギガビット・イーサネットおよびそれより高速なアダプターのパフォーマンスが、このオプションにより向上します。ただし、仮想イーサネット・アダプターおよび共用イーサネット・アダプター (SEA) デバイスは例外です。これらのデバイスでは、Linux または IBM i オペレーティング・システムとの相互運用性に問題があるため、大容量送信オフロード・オプションはデフォルトで使用不可になっています。 AIXおよび仮想イーサネット・アダプターまたは SEA 環境では、大容量送信およびその他のパフォーマンス・フィーチャーを使用可能にすることができます。

大容量送信オプションは、large_send で指定されるデバイス属性です。大容量送信オフロード・デバイス属性を表示するには、次のコマンドを使用してください。X はデバイス番号です。

```
lsattr -E -l entX
```

アダプター・オフロードのオプション*:*

一部のアダプターには、AIX システムからアダプターへのオフロード作業を使用可能または使用不可にするオプションがあります。

表 *7.* アダプターと使用可能なオプション、およびシステム・デフォルト設定

アダプター・タイプ	フィーチャー・コード	**TCP** チェックサム・オフロード	デフォルト設定値	**TCP** 大容量送信	デフォルト設定値
GigE、PCI、SX および TX	2969, 2975	はい	オフ	はい	オフ
GigE、PCI-X、SX および TX	5700, 5701	はい	ON	はい	ON
GigE デュアル・ポート PCI-X、TX および SX	5706, 5707	はい	ON	はい	ON
10 GigE PCI-X LR および SR	5718, 5719	はい	ON	はい	ON
10/100 イーサネット	4962	はい	ON	はい	オフ
ATM 155、UTP および MMF	4953, 4957	はい (送信のみ)	ON	いいえ	N/A
ATM 622、MMF	2946	はい	ON	いいえ	N/A

TCP 要求/応答ワークロードのチューニング

TCP 要求/応答ワークロードは、両方向の情報交換を伴うワークロードです。

要求および応答ワークロードの例としては、リモート・プロシージャー・コール (RPC) タイプのアプリケーションやクライアント/サーバー・アプリケーションがあります。 Web サーバーへの Web ブラウザー要求、NFS ファイルシステム (トランスポート・プロトコルとして TCP を使用)、データベースのロック管理プロトコルなどがこれにあたります。 このような要求は、メッセージが小さく、応答が大きくなることが多いですが、要求が大きく、応答が小さくなる場合もあります。

これらのワークロードで重要な意味を持つ測定基準は、ネットワークの往復待ち時間です。 この種の要求や応答は、小さなメッセージを使用する場合が多いので、ネットワーク帯域幅は大きな問題になりません。

ハードウェアは待ち時間に大きな影響を与えます。 例えば、ネットワークのタイプ、ネットワーク・スイッチやルーターのタイプとパフォーマンス、ネットワークの各ノードで使用するプロセッサーの速度、アダプターやバスの待ち時間はいずれも、往復時間に影響を与えます。

一般に、待ち時間を最小にする (応答速度を最速にする) チューニング・オプションを設定すると、システムが待ち時間と応答時間を最小にするためにより多くのパケットを送信するとか、より多くの割り込みを処理するといったことが必要になるため、CPU のオーバーヘッドが大きくなります。 これは、パフォーマンスに関する古典的なトレードオフです。

要求/応答アプリケーションの主なチューナブルなものは、以下のとおりです。

- *tcp_nodelay* または *tcp_nagle_limit*
- *tcp_nodelayack*
- アダプター割り込み合体設定

注**:** 一部の要求/応答ワークロードでは、1 方向のデータが大量になります。 このようなワークロードは、ワークロードごとに、ストリーミングと待ち時間を組み合わせてチューニングすることが必要になります。

tcp_nodelay または **tcp_nagle_limit** オプション**:**

AIX では、TCP_NODELAY ソケット・オプションはデフォルトで使用不可になっているため、要求/応答ワークロードにかなりの遅延が生じることがあり、数バイトだけ送信しては応答を待つこともあります。 TCP には、TCP 肯定応答が応答パケットに「背負われて」伝送されるよう期待して、遅延応答がインプリメントされています。 遅延は通常 200 ミリ秒です。

ほとんどの TCP インプリメンテーションでは Nagle アルゴリズムがインプリメントされますが、このアルゴリズムでは 1 つの TCP 接続に未応答の小さい未解決セグメントを 1 つしか取ることができません。 そのため、肯定応答を受信するか、またはいくつかのデータをまとめてフルサイズのセグメントとして送信できるようになるまで、TCP はパケットの送信を遅延します。

要求/応答ワークロードを使用するアプリケーションでは、**setsockopt()** コールを使用して TCP_NODELAY オプションを有効にするようにします。 例えば、**telnet** および **rlogin** ユーティリティー、ネットワーク・ファイルシステム (NFS)、および Web サーバーでは、Nagle を使用不可にするために、既に TCP_NODELAY オプションが使用されています。 しかし、一部のアプリケーションではそうなっていないため、ネットワーク MTU サイズとソケットへの送信 (書き込み) サイズによっては、パフォーマンスが低下します。

TCP_NODELAY を有効にしないアプリケーションを扱うときには、以下のチューニング・オプションを使用して Nagle を使用不可にすることができます。

- **tcp_nagle_limit**
- **tcp_nodelay** ISNO オプション
- **tcp_nodelayack**

- **fasttimo**
- アダプターでの割り込みの合体

tcp_nagle_limit オプション*:*

tcp_nagle_limit ネットワーク・オプションはグローバル・ネットワーク・オプションで、デフォルトでは 65536 に設定されています。

TCP はこの値と等しいまたはそれより大きいセグメントに対して Nagle アルゴリズムを使用不可にし、Nagle が使用可能となるしきい値を調整できるようにします。 例えば、Nagle を完全に使用不可にするには、*tcp_nagle_limit* 値を 1 に設定します。 TCP が送信をまとめて、少なくとも 256 バイトになってからパケットを送信できるようにするには、*tcp_nagle_limit* 値を 256 に設定します。

tcp_nodelay ISNO オプション*:*

インターフェース・レベルでは、TCP_NODELAY を有効にする *tcp_nodelay* ISNO オプションがあります。

tcp_nodelay 値を 1 に設定すると、TCP は遅延せず、アプリケーションが送信や書き込みを実行するたびに各パケットを送信します。

tcp_nodelayack オプション*:*

tcp_nodelayack ネットワーク・オプションを使用して、遅延肯定応答 (通常は 200 ミリ秒タイマー) を使用不可にすることができます。

肯定応答を遅延させないことにより、待ち時間が少なくなるとともに、送信側 (Nagle が有効になっている場合がある) は肯定応答を受信してすぐに次の部分セグメントを送信することができます。

fasttimo オプション*:*

fasttimo ネットワーク・オプションを使用すると、200 ミリ秒 (デフォルト) のタイマーを 100 または 50 ミリ秒に減らすことができます。

TCP は開いているすべての TCP 接続に対して他の機能を実行する際にもこのタイマーを使用するので、このタイマーを減らすと、すべての TCP 接続をより頻繁にスキャンしなければならなくなって、システムのオーバーヘッドがさらに増えます。 実際に選択する場合は、前述の他のオプションを使用するのが最善です。**fasttimo** オプションは、システム調整の最後の手段としてのみ使用してください。

割り込みの合体*:*

ホスト・システムへの過大な割り込み発生を回避するには、パケットを集めて複数のパケットに対して 1 つの割り込みが生成されるようにします。 これは「割り込み合体」と呼ばれます。

一般に受信操作での割り込みは、パケットがデバイスの入力キューに到着したことをホスト CPU に通知するものです。 アダプターに何らかの形式の割り込み緩和ロジックがないと、着信パケットごとに割り込みが発生する可能性があります。 ただし、着信パケットの速度が上がると、デバイス・ドライバーは 1 パケットの処理を終えた後に、受信キューにこれ以上パケットがないことを確認してから、ドライバーの終了と割り込みのクリアを行います。 パケットの速度が上がってドライバーが処理対象パケットがまだあることを検出すると、ドライバーは割り込みごとに複数のパケットを処理するようになります。このことは、負荷が増すとシステムの効率が向上することを意味します。

一方、アダプターによっては、受信割り込みをいつ生成するかについても制御できる追加機能があるものもあります。 これはしばしば割り込み合体または割り込み緩和ロジックと呼ばれ、この機能を使用することで、いくつかのパケットを受信したり、いくつかのパケットで 1 つの割り込みを生成したりすることが可能です。 最初のパケットが到着するとタイマーが開始され、割り込みは *n* マイクロ秒間または *m* パケットが到着するまで遅延させられます。 その方式はアダプターによって異なり、またデバイス・ドライバーがどの機能についてユーザーが制御できるようにしているかによっても異なります。

負荷が軽い場合の割り込み合体では、待ち時間がパケット到着時刻に追加されます。 パケットはホスト・メモリーにありますが、ホストはしばらくの間はパケットに気付きません。 一方、パケット負荷が大きい場合は、生成される割り込みが少なくなり、ホストは割り込みごとにいくつかのパケットを処理するために使用 CPU サイクルが少なくてすむので、システムの実行効率が上がります。

割り込み緩和機能が組み込まれた AIX アダプターでは、長い待ち時間が追加されずに割り込みオーバーヘッドが減少する、適度のレベルに値を設定する必要があります。 最小待ち時間が必要になる可能性のあるアプリケーションについては、待ち時間を短くするために、オプションを使用不可にするか変更して、1 秒当たりの可能な割り込みを増やす必要があります。

ギガビット・イーサネット・アダプターには、割り込み緩和機能があります。 FC 2969 および FC 2975 GigE PCI アダプターでは、遅延値とバッファー・カウントの方式を使用できます。 アダプターは最初のパケットが到着するとタイマーを開始し、タイマーの時間が経過するかホストで *n* 個のバッファーが使用されると、割り込みが発生します。

FC 5700、FC 5701、FC 5706、および FC 5707 GigE PCI-X アダプターでは、割り込みスロットル頻度方式が使用されます。この方式では指定した頻度で割り込みが生成されます。したがって、この方式では時間を基準にしてパケットを集めることができます。 デフォルトの割り込み頻度は、1 秒当たり 10 000 割り込みです。割り込みオーバーヘッドを減らすには、割り込み頻度を最小の 1 秒当たり 2 000 割り込みに設定します。 短い待ち時間と速い応答時間を必要とするワークロードの場合は、割り込み頻度を最大の 20 000 割り込みに設定します。 割り込み頻度を 0 に設定すると、割り込みスロットルは完全に使用不可になります。

10 Gigabit Ethernet PCI-X アダプター (FC 5718 and 5719) には、0.82 マイクロ秒の遅延装置で指定できる割り込み合体オプション (*rx_int_delay*) があります。 実際の遅延時間は *rx_int_delay* に設定された値に 0.82 を乗算して決定されます。このオプションは、テストの結果、これらのアダプターの入力速度が高いと、割り込みの合体でパフォーマンスを向上させられないことが判り、デフォルトでは使用不可に設定されています (*rx_int_delay*=0)。

表 *8. 10 Gigabit Ethernet PCI-X* アダプターの特性

アダプター・タイプ	フィーチャー・コード	**ODM** 属性	デフォルト値	範囲
10 Gigabit Ethernet PCI-X (LR または SR)	5718, 5719	rx_int_delay	0	0-512

UDP チューニング

ユーザー・データグラム・プロトコル (UDP) は、ネットワーク・ファイルシステム (NFS)、ネーム・サーバー (named)、トリビアル・ファイル転送プロトコル (TFTP)、およびその他の特殊な用途のプロトコルで使用されるデータグラム・プロトコルです。

UDP はデータグラム・プロトコルなので、送信操作時に 1 つのアトミック操作として、メッセージ (データグラム) 全体をカーネルにコピーする必要があります。 データグラムは、 **recv** または **recvfrom** システム・コールでも、1 つの完全なメッセージとして受信されます。 ソケットごとに、バッファリング要件を処理するための *udp_sendspace* および *udp_recvspace* パラメーターを設定する必要があります。

送信可能な最大 UDP データグラムは、64 KB から UDP ヘッダー・サイズ (8 バイト) および IP ヘッダー・サイズ (IPv4 で 20 バイト、IPv6 で 40 バイト) を引いた値になります。

UDP パフォーマンスに影響を与えるチューニング・オプションは、以下のとおりです。

- *udp_sendspace*
- *udp_recvspace*
- UDP パケット・チェーニング
- 割り込み合体などのアダプター・オプション

udp_sendspace チューニング・オプション:

udp_sendspace チューニング・オプション値は、送信する最大の UDP データグラム以上の値に設定します。

わかりやすくするために、このパラメーターは 65536 に設定してください。この値は UDP パケットが取りうる最大サイズを処理するにも十分な大きさです。 この値をこれ以上大きく設定しても効果はありません。

udp_recvspace チューニング・オプション:

udp_recvspace チューニング値は、各 UDP ソケットのキューに入れられる着信データのためのスペースの量を制御します。 ソケットが *udp_recvspace* の限界に達すると、着信するパケットは破棄されます。

破棄されたパケットの統計は、 **netstat -p udp** コマンド出力の「`socket buffer overflows` (ソケット・バッファーがオーバーフロー)」欄に詳しく記述されます。詳しくは、*Commands Reference, Volume 4*の『netstat コマンド』を参照してください。

複数の UDP データグラムが送信され、アプリケーションがそのデータグラムを読み取るまでソケットで待機することもあるので、*udp_recvspace* チューニング値は大きい値に設定してください。 また、多数の UDP アプリケーションはパケットの受信に特定のソケットを使用します。 このソケットは、サーバー・アプリケーションと通信するすべてのクライアントからパケットを受信するために使用されます。 そのため受信スペースには、複数のクライアントから一度に送信される大量のデータグラムを処理し、読み取りを待つ間ソケットでキューに入れることが可能な十分の大きさが必要です。 この値が小さすぎると、着信パケットは破棄されてしまい、送信側はパケットを再送しなければなりません。 結果として、パフォーマンスは低下します。

通信サブシステムは、バッファーの内容ではなく、使用されているバッファーを算出するので、*udp_recvspace* の設定時にこれを算出する必要があります。例えば、8 KB のデータグラムが 6 つのパケットにフラグメント化された場合、6 つの受信バッファーが使用されることになります。 これらはイーサネットの場合は、2048 バイトのバッファーになります。 そうすると、この 1 つの 8 KB データグラムが使用するソケット・バッファーの合計量は、次のとおりです。

```
6*2048=12,288 bytes
```

したがって、着信バッファリングの効率に応じて、*udp_recvspace* を大きい値に調整する必要があるかどうかが分かります。 これは、データグラム・サイズとデバイス・ドライバーによって異なります。 64 バイトのデータグラムを送信すると、64 バイトの各データグラムごとに 2 KB バッファーが使用されます。

このあと、この 1 つのソケットのキューに入れられる可能性のあるデータグラムの数を算出する必要があります。 例えば、NFS サーバーは、すべてのクライアントから予約済みソケットに UDP パケットを受

け取ります。 このソケットのキュー項目数が 30 パケットである場合、NFS が 8 KB データグラムを使用するときには、*udp_recvspace* の計算が 30 *12,288 = 368,640 となります。 NFS バージョン 3 では、32KB までのデータグラムが認められます。

UDP はもう 1 つのパケットが到着するまでパケットをアプリケーションに渡すことができない場合があるため、*udp_recvspace* の推奨開始値は、*udp_sendspace* の値の 10 倍です。また、同時に幾つかのノードを 1 つのノードに送信できます。 ステージング・スペースを提供するには、後続のパケットが廃棄される前に、10 個のパケットがステージングされるように、このサイズが設定されます。 UDP を使用する大規模な並列アプリケーションの場合は、値を大きくする必要があります。

注**:** ソケット・バッファーの最大ソケット・バッファー・サイズを指定する *sb_max* の値は、UDP および TCP の送信バッファーと受信バッファーの最大サイズを 2 倍以上にした値でなければなりません。

UDP パケット・チェーニング**:**

送信される UDP データグラムがアダプターの MTU サイズよりも大きい場合、 IP プロトコル層はデータグラムを MTU サイズのフラグメントに分割します。イーサネット・インターフェースには、UPD パケット・チェーニング機能が組み込まれています。 この機能は、 AIX ではデフォルトで使用可能です。

UDP パケット・チェーニングとは、IP がフラグメントのチェーン全体を構築し、そのチェーンをイーサネット・デバイス・ドライバーに 1 つのコールで渡すという機能です。 この機能を使用すれば、ARP 層とインターフェース層を経由してドライバーに達するコールの数を削減できるので、パフォーマンスが向上します。 さらに SMP 環境においても、**lock** および **unlock** コールの数を削減できます。 これは、コード・ループのキャッシュ親和性に役立ちます。 また、こうした変化によって、送信側の CPU 使用率は下がります。

UDP パケット・チェーニング・オプションは、**ifconfig** コマンドで確認できます。以下の例では、`en0` インターフェースに関する **ifconfig** コマンド出力を表示します。この場合、`CHAIN` フラグは、パケット・チェーニングが使用可能であることを示します。

```
# ifconfig en0
en0: flags=5e080863,80<UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST,GROUPRT,64BIT,CHECKSUM_OFFLOAD,PSEG,CHAIN>
         inet 192.1.6.1 netmask 0xffffff00 broadcast 192.1.6.255
         tcp_sendspace 65536 tcp_recvspace 65536 tcp_nodelay 1
```

パケット・チェーニングを使用不可にするには、以下のコマンドを使用します。

```
# ifconfig en0 -pktchain

# ifconfig en0
en0: flags=5e080863,80<UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST,GROUPRT,64BIT,CHECKSUM_OFFLOAD,PSEG>
         inet 192.1.6.1 netmask 0xffffff00 broadcast 192.1.6.255
         tcp_sendspace 65536 tcp_recvspace 65536 tcp_nodelay 1
```

パケット・チェーニングを再び使用可能にするには、以下のコマンドを使用します。

```
# ifconfig en0 pktchain

# ifconfig en0
en0: flags=5e080863,80<UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST,GROUPRT,64BIT,CHECKSUM_OFFLOAD,PSEG,CHAIN>
         inet 192.1.6.1 netmask 0xffffff00 broadcast 192.1.6.255
         tcp_sendspace 65536 tcp_recvspace 65536 tcp_nodelay 1
```

割り込みの合体**:**

ホスト・システムへの過大な割り込み発生を回避するには、パケットを集めて複数のパケットに対して 1 つの割り込みが生成されるようにします。 これは「割り込み合体」と呼ばれます。

一般に受信操作での割り込みは、パケットがデバイスの入力キューに到着したことをホスト CPU に通知するものです。 アダプターに何らかの形式の割り込み緩和ロジックがないと、着信パケットごとに割り込

みが発生する可能性があります。 ただし、着信パケットの速度が上がると、デバイス・ドライバーは 1 パケットの処理を終えた後に、受信キューにこれ以上パケットがないことを確認してから、ドライバーの終了と割り込みのクリアを行います。 パケットの速度が上がってドライバーが処理対象パケットがまだあることを検出すると、ドライバーは割り込みごとに複数のパケットを処理するようになります。このことは、負荷が増すとシステムの効率が向上することを意味します。

一方、アダプターによっては、受信割り込みをいつ生成するかについても制御できる追加機能があるものもあります。 これはしばしば割り込み合体または割り込み緩和ロジックと呼ばれ、この機能を使用することで、いくつかのパケットを受信したり、いくつかのパケットで 1 つの割り込みを生成したりすることが可能です。 最初のパケットが到着するとタイマーが開始され、割り込みは *n* マイクロ秒間または *m* パケットが到着するまで遅延させられます。 その方式はアダプターによって異なり、またデバイス・ドライバーがどの機能についてユーザーが制御できるようにしているかによっても異なります。

負荷が軽い場合の割り込み合体では、待ち時間がパケット到着時刻に追加されます。 パケットはホスト・メモリーにありますが、ホストはしばらくの間はパケットに気付きません。 一方、パケット負荷が大きい場合は、生成される割り込みが少なくなり、ホストは割り込みごとにいくつかのパケットを処理するために使用 CPU サイクルが少なくてすむので、システムの実行効率が上がります。

割り込み緩和機能が組み込まれた AIX アダプターでは、長い待ち時間が追加されずに割り込みオーバーヘッドが減少する、適度のレベルに値を設定する必要があります。 最小待ち時間が必要になる可能性のあるアプリケーションについては、待ち時間を短くするために、オプションを使用不可にするか変更して、1 秒当たりの可能な割り込みを増やす必要があります。

ギガビット・イーサネット・アダプターには、割り込み緩和機能があります。 FC 2969 および FC 2975 GigE PCI アダプターでは、遅延値とバッファー・カウントの方式を使用できます。 アダプターは最初のパケットが到着するとタイマーを開始し、タイマーの時間が経過するかホストで *n* 個のバッファーが使用されると、割り込みが発生します。

FC 5700、FC 5701、FC 5706、および FC 5707 GigE PCI-X アダプターでは、割り込みスロットル頻度方式が使用されます。この方式では指定した頻度で割り込みが生成されます。したがって、この方式では時間を基準にしてパケットを集めることができます。 デフォルトの割り込み頻度は、1 秒当たり 10 000 割り込みです。割り込みオーバーヘッドを減らすには、割り込み頻度を最小の 1 秒当たり 2 000 割り込みに設定します。 短い待ち時間と速い応答時間を必要とするワークロードの場合は、割り込み頻度を最大の 20 000 割り込みに設定します。 割り込み頻度を 0 に設定すると、割り込みスロットルは完全に使用不可になります。

10 Gigabit Ethernet PCI-X アダプター (FC 5718 and 5719) には、0.82 マイクロ秒の遅延装置で指定できる割り込み合体オプション (*rx_int_delay*) があります。 実際の遅延時間は *rx_int_delay* に設定された値に 0.82 を乗算して決定されます。このオプションは、テストの結果、これらのアダプターの入力速度が高いと、割り込みの合体でパフォーマンスを向上させられないことが判り、デフォルトでは使用不可に設定されています (*rx_int_delay*=0)。

表 9. 10 Gigabit Ethernet PCI-X アダプターの特性

アダプター・タイプ	フィーチャー・コード	**ODM** 属性	デフォルト値	範囲
10 Gigabit Ethernet PCI-X (LR または SR)	5718, 5719	rx_int_delay	0	0-512

アダプター・リソースのチューニング

アダプターとドライバーは広範囲にわたるため、すべてのタイプのアダプターの属性を取り上げることは困難です。 以下の情報は、大半のネットワーク・アダプターおよびドライバーにあり、システム・パフォーマンスに影響する一般的な属性に焦点を当てたものです。

多くの通信ドライバーには、送信と受信のリソースを制御できる一連のチューナブル・パラメーターが用意されています。 これらのパラメーターは一般的に、送信キューと受信キューの制限を制御しますが、バッファーまたはその他のリソースの数とサイズを制御することもできます。 これらのパラメーターは、送信キューに入れることができるバッファーまたはパケットの数を制限したり、受信パケットに使用できる受信バッファーの数を制限したりします。 これらのパラメーターは、システムまたはネットワークから生成されるピーク・ロードを処理するのに十分な、アダプター・レベルのキューイングを可能にするために調整できます。

以下は、一般的なガイドラインです。

- アダプター・リソースおよび発生したエラーに関する詳細情報を表示するには、ご使用のアダプターに応じて、以下のコマンドを使用します。
  - **netstat -v**
  - **entstat**
  - **atmstat**
  - **fddistat**
  - **tokstat**
- システム・エラー・ログ・レポートをモニターするには、**errpt** および **errpt -a** コマンドを使用します。
- 以下のいずれかの状態が該当する場合は、忘れずにパラメーターのみを変更してください。
  - リソース不足を示す兆候がある。
  - キューのオーバーランがある。
  - パフォーマンス分析で、システムの何らかのチューニングが必要であることが分かっている。

送信キュー**:**

送信の場合、デバイス・ドライバーには送信キュー 制限を規定するものがあります。

ドライバーとアダプターによっては、ハードウェア・キュー制限とソフトウェア・キュー制限の両方があります。 ドライバーのなかには、ハードウェア・キューだけが存在するものと、ハードウェアとソフトウェアの両方のキューが存在するものがあります。 また、ハードウェア・キューを内部的に制御し、変更できるのはソフトウェア・キュー制限のみというものもあります。 一般的には、デバイス・ドライバーが送信パケットを直接アダプターのハードウェア・キューに入れます。 システム CPU がネットワーク速度と比較して高速である場合、または SMP システムでは、システムがネットワークで送信できるよりも高速で送信パケットを生成することがあります。 この場合は、ハードウェア・キューが満杯になります。

ハードウェア・キューが満杯になると、ドライバーによってはソフトウェア・キューを提供するものもあるので、そのソフトウェア・キューに入れられます。 ソフトウェア送信キュー制限に達すると、送信パケッ

トは廃棄されます。これにより、上位プロトコルがパケットをタイムアウトにして再送するため、パフォーマンスが影響を受けることがあります。 ただし過度のスペースが指定されると、不整合パケットが送信されることがあるので、このような場合はアダプターはある時点でパケットを廃棄する必要があります。

表 *10. PCI* アダプター送信キュー・サイズの例

アダプター・タイプ	フィーチャー・コード	**ODM** 属性	デフォルト値	範囲
IBM 10/100 Mbps Ethernet PCI アダプター	2968	tx_que_size	8192	16 から 16384
10/100 Mbps Ethernet Adapter II	4962	tx_que_sz	8192	512 から 16384
Gigabit Ethernet PCI (SX または TX)	2969, 2975	tx_que_size	8192	512 から 16384
Gigabit Ethernet PCI (SX または TX)	5700, 5701, 5706, 5707	tx_que_sz	8192	512 から 16384
10 Gigabit Ethernet PCI-X (LR または SR)	5718, 5719	tx_que_sz	8192	512 から 16384
ATM 155 (MMF または UTP)	4953, 4957	sw_txq_size	2048	50 から 16384
ATM 622 (MMF)	2946	sw_txq_size	2048	128 から 32768
FDDI	2741, 2742, 2743	tx_queue_size	256	3 から 2048

ハードウェア・キュー制限を規定するアダプターの場合は、これらの値を変更すると、関連付けられている制御ブロックとそれらに関連付けられているバッファーが原因で、受信時により多くの実メモリーが消費されます。 したがって、必要な場合、またはメモリー使用量の増加がわずかであるより大きいシステムの場合にのみ、これらの制限を大きくしてください。 ソフトウェア送信キュー制限の場合は、これらの制限を大きくしても、メモリーの使用率は増えません。 高位のレイヤー・プロトコルが既に割り当てているパケットをキューに入れることができるようになるだけです。

送信ディスクリプター**:**

ドライバーによっては、送信リングのサイズまたは送信ディスクリプターの数をチューニングできるものもあります。

ハードウェア送信キューは、同時送信の場合にアダプターのキューに入れることができるバッファーの最大数を制御します。 通常は 1 つのディスクリプターは 1 つのバッファーのみを指し、メッセージは複数のバッファーに送信されることがあります。 多くのドライバーで、ユーザーはこのパラメーターを変更できないようになっています。

アダプター・タイプ	フィーチャー・コード	**ODM** 属性	デフォルト値	範囲
Gigabit Ethernet PCI-X、SX、または TX	5700, 5701, 5706, 507	txdesc_que_sz	512	128 から 1024 (128 の倍数)

受信リソース**:**

アダプターによっては、ネットワークから受信パケットに使用するリソースの数を構成できるものもあります。 これには、受信バッファーの数 (およびそのサイズ) または DMA 受信ディスクリプターの数が含まれることがあります。

ドライバーによっては、バッファーのサイズが異なる複数の受信バッファー・プールを持つものもあり、ワークロードが異なるとこれをチューニングする必要がある場合があります。一部のドライバーはこれらのリソースをドライバー内部で管理し、ユーザーはそれらを変更できないようになっています。

ネットワーク上のピーク・バーストを処理するために、受信リソースを増やす必要のある場合があります。ネットワーク・インターフェースのデバイス・ドライバーは、受信キューに着信パケットを入れます。 受

信ディスクリプター・リストまたはリングが満杯になった場合、または使用可能なバッファーがなくなった場合は、パケットが破棄されるため、送信側による再送が必要になります。 受信ディスクリプター・キューのチューニングは、SMIT ツールまたは **chdev** コマンドを使用して行うことができます (311 ページの『ネットワーク・パラメーターの変更』を参照してください)。 最大キュー・サイズは通信アダプターのそれぞれのタイプごとに固有であり、通常は、SMIT ツールの **F4** or **List** キーを使用して表示できます。

表 *11. PCI* アダプター受信キュー・サイズの例

アダプター・タイプ	フィーチャー・コード	**ODM** 属性	デフォルト値	範囲
IBM 10/100 Mbps Ethernet PCI アダプター	2968	rx_que_size rx_buf_pool_size	256 384	16, 32 ,64, 128, 26 16 から 2048
10/100 Mbps Ethernet PCI Adapter II	4962	rx_desc_que_sz rxbuf_pool_sz	512 1024	100 から 1024 512 から 2048
Gigabit Ethernet PCI (SX または TX)	2969, 2975	rx_queue_size	512	512 (固定)
Gigabit Ethernet PCI-X (SX または TX)	5700、 5701、 5706、 5707、 5717、 5768、 5271、 5274 、 5767、 および 5281	rxbuf_pool_sz rxdesc_que_sz	2048 1024	512-16384,1 128-3840,128
10 Gigabit PCI-X (SR または LR)	5718, 5719	rxdesc_que_sz rxbuf_pool_sz	1024 2048	128 から 1024 (128 単位) 512 から 2048
ATM 155 (MMF または UTP)	4953, 4957	rx_buf4k_min	x60	x60 から x200 (96 から 512)
ATM 622 (MMF)	2946	rx_buf4k_min rx_buf4k_max	256 [2] 0 [1]	0 から 4096 0 から 14000
FDDI	2741, 2742, 2743	RX_buffer_cnt	42	1 から 512

注**:**

1. ATM アダプターの **rx_buf4k_max** 属性は、受信バッファー・プール内のバッファーの最大数です。 値を 0 に設定すると、ドライバーはシステムのメモリー量に基づいて数を割り当てます (例えば、**rx_buf4k_max= thewall** * 6 / 320)。ただし、ATM 155 アダプターの場合の上限は 9500 バッファー、ATM 622 アダプターの場合の上限は 16360 バッファーです。 必要のなくなったバッファーは解放されます (**rx_buf4k_min** まで)。
2. ATM アダプターの **rx_buf4k_min** 属性は、プール内のフリー・バッファーの最小数です。 ドライバーは、この量だけのフリー・バッファーをプール内に保持しようとします。 プールは最大で **rx_buf4k_max** の値まで拡張されます。

デバイス属性の照会および変更のコマンド**:**

いくつかの状況ユーティリティーを使用して、送信キューの上限と、「*no resource*」または「*no buffer*」エラーの数を表示することができます。

netstat -v コマンドを使用することもできますし、アダプター統計情報ユーティリティー (イーサネットの場合は **entstat**、トークンリングの場合は **tokstat**、FDDI の場合は **fddistat**、ATM の場合は **atmstat** など) に直接移動することもできます。

entstat の出力例については、 353 ページの『アダプター統計情報』を参照してください。もう 1 つの方法は、**netstat -i** ユーティリティーを使用する方法です。インターフェースの「*Oerrs*」欄にゼロ以外のカウントが表示された場合は、出力キュー・オーバーフローの結果であることが一般的です。

ネットワーク・アダプター設定の表示**:**

アダプター構成を表示するには、**lsattr -E -l** *adapter-name* コマンドを使用することもできますし、SMIT コマンド (**smitty commodev**) を使用することもできます。

タイプの異なるアダプターは、これらの変数の名前が異なります。例えば、送信キュー・パラメーターに *sw_txq_size*、*tx_que_size*、または *xmt_que_size* などの名前が付けられていることがあります。例えば、受信キュー・サイズと受信バッファー・プールのパラメーターには、*rec_que_size*、*rx_que_size*、または *rv_buf4k_min* などの名前が付けられています。

以下は、IBM PCI 622 Mbps ATM アダプターについての **lsattr -E -l atm0** コマンドの出力例です。 この出力から、*sw_txq_size* が 2048 に設定されており、*rx_buf4K_min* 受信バッファーが 256 に設定されていることが分かります。

```
# lsattr -E -l atm0
adapter_clock  0           Provide SONET Clock                           True
alt_addr       0x0         ALTERNATE ATM MAC address (12 hex digits)     True
busintr        99          Bus Interrupt Level                           False
interface_type 0           Sonet or SDH interface                        True
intr_priority  3           Interrupt Priority                            False
max_vc         1024        Maximum Number of VCs Needed                  True
min_vc         64          Minimum Guaranteed VCs Supported              True
regmem         0xe0008000  Bus Memory address of Adapter Registers       False
rx_buf4k_max   0           Maximum 4K-byte pre-mapped receive buffers    True
rx_buf4k_min   256         Minimum 4K-byte pre-mapped receive buffers    True
rx_checksum    yes         Enable Hardware Receive Checksum              True
rx_dma_mem     0x4000000   Receive  bus memory address range             False
sw_txq_size    2048        Software Transmit Queue size                  True
tx_dma_mem     0x2000000   Transmit bus memory address range             False
uni_vers       auto_detect SVC UNI Version                               True
use_alt_addr   no          Enable ALTERNATE ATM MAC address              True
virtmem        0xe0000000  Bus Memory address of Adapter Virtual Memory False
```

以下は、**lsattr -E -l ent0** コマンドを使用して出力した PCI-X Gigabit Ethernet アダプターの設定の例です。この出力から、*tx_que_size* が 8192 に、*rxbuf_pool_sz* が 2048 に、*rx_que_size* が 1024 に、それぞれ設定されていることが分かります。

```
# lsattr -E -l ent0

alt_addr       0x000000000000   Alternate ethernet address                  True
busintr        163              Bus interrupt level                         False
busmem         0xc0080000       Bus memory address                          False
chksum_offload yes              Enable hardware transmit and receive checksum True
compat_mode    no               Gigabit Backward compatibility              True
copy_bytes     2048             Copy packet if this many or less bytes      True
flow_ctrl      yes              Enable Transmit and Receive Flow Control    True
intr_priority  3                Interrupt priority                          False
intr_rate      10000            Max rate of interrupts generated by adapter True
jumbo_frames   no               Transmit jumbo frames                       True
large_send     yes              Enable hardware TX TCP resegmentation       True
media_speed    Auto_Negotiation Media speed                                 True
rom_mem        0xc0040000       ROM memory address                          False
rx_hog         1000             Max rcv buffers processed per rcv interrupt True
rxbuf_pool_sz  2048             Rcv buffer pool, make 2X rxdesc_que_sz      True
rxdesc_que_sz  1024             Rcv descriptor queue size                   True
slih_hog       10               Max Interrupt events processed per interrupt True
```

```
tx_que_sz      8192             Software transmit queue size                True
txdesc_que_sz  512              TX descriptor queue size                    True
use_alt_addr   no               Enable alternate ethernet address           True
```

ネットワーク・パラメーターの変更

ネットワーク・パラメーターを変更する場合は、可能なら常に **smitty** コマンドを使用してください。

特定のデバイス・タイプを選択するには、**smitty commodev** コマンドを使用します。そこで表示されるリストからアダプター・タイプを選択します。 以下は、**smitty commodev** コマンドでイーサネット・アダプターのネットワーク・パラメーターを変更する場合の例です。

```
                        Change/Show Characteristics of an Ethernet Adapter

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                [Entry Fields]
  Ethernet Adapter                             ent2
  Description                                  10/100/1000 Base-TX PCI-X Adapter (14106902)
  Status                                       Available
  Location                                     1V-08
  Receive descriptor queue size               [1024]                                          +#
  Transmit descriptor queue size              [512]                                           +#
  Software transmit queue size                [8192]                                          +#
  Transmit jumbo frames                        no                                             +
  Enable hardware transmit TCP resegmentation  yes                                            +
  Enable hardware transmit and receive checksum yes                                           +
  Media Speed                                  Auto_Negotiation                               +
  Enable ALTERNATE ETHERNET address            no                                             +
  ALTERNATE ETHERNET address                  [0x000000000000]                                +
  Apply change to DATABASE only                no                                             +

F1=Help                    F2=Refresh                 F3=Cancel                  F4=List
Esc+5=Reset                Esc+6=Command              Esc+7=Edit                 Esc+8=Image
Esc+9=Shell                Esc+0=Exit                 Enter=Do
```

パラメーター値を変更するには、次のようにします。

1. 次のコマンドを実行して、インターフェースを切り離します。

   ```
   # ifconfig en0 detach
   ```

   この *en0* はアダプター名です。
2. SMIT を使用して、アダプターの設定を表示します。 「**Devices (**デバイス**)**」**->**「**Communications(**通信**)**」**->**「*adapter type* (アダプター・タイプ)」**->**「**Change/Show... (**変更**/**表示**)**」を選択します。
3. 変更したいフィールドへカーソルを移動し、**F4** キーを押して、フィールド (またはサポートされているサイズの特定のセット) の最小および最大範囲を確認します。
4. 適切なサイズを選択し、Enter キーを押して、ODM データベースを更新します。
5. 次のコマンドを実行して、アダプターを再接続します。

   ```
   # ifconfig en0 hosthame up
   ```

これらのパラメーター値を変更するもう 1 つの方法は、次のコマンドを実行する方法です。

```
# chdev -l [ifname] -a [attribute-name]=newvalue
```

例えば、en0 上の上記の **tx_que_size** を 128 に変更するには、次の一連のコマンドを使用します。 このドライバーは 4 つのサイズだけをサポートしているため、値を確認するには SMIT コマンドを使用した方がよいことに注意してください。

```
# ifconfig en0 detach
# chdev -l ent0 -a tx_que_size=128
# ifconfig en0 hostname up
```

TCP 最大セグメント・サイズのチューニング

ネットワーク上で可能な限り大きなサイズのパケットを送信するほうが効率的なので、TCP 送信のパケット最大サイズは、帯域幅に大きな影響を与える可能性があります。

TCP は TCP 接続ごとに、この最大サイズを制御します。このサイズのことを最大セグメント・サイズ (MSS) といいます。 直接接続のネットワークの場合、TCP は、ネットワーク・インターフェースの MTU サイズからプロトコル・ヘッダーのサイズを差し引いて MSS サイズを計算し、TCP パケット内のデータ・サイズを導き出します。 例えば、MTU が 1500 のイーサネットでは、 20 バイトの IPv4 ヘッダーと 20 バイトの TCP ヘッダーを差し引いた 1460 が MSS サイズになります。

TCP プロトコルには、接続の作成時に、その接続上で使用される MSS を接続の両端で通知するための機構が組み込まれています。両端ではそれぞれ、TCP ヘッダー内の OPTIONS フィールドを使用して、提示された MSS を通知します。 選択される MSS は、両端が提供した値の小さい方です。 片方の端点が MSS を提供しない場合は、536 バイトが想定され、パフォーマンスの低下が発生します。

ここでの問題は、それぞれの TCP 端点が接続先のネットワークの MTU しか知らないということです。2 つの端点の間に存在する可能性のある他のネットワークの MTU サイズは分かりません。 したがって、両方の端点が同一のネットワーク上に存在する場合にのみ、TCP は正しい MSS を知ることになります。そのため TCP は、MTU の小さいネットワーク上で IP フラグメント化が必要になるようなパケット送信を回避する場合には、ネットワークの構成に基づいて MSS の通知を別々の方法で処理します。

接続セットアップ時に TCP ソフトウェアによって通知される MSS の値は、もう一方の接続端が同じ物理ネットワーク上のローカル・システムであるか (つまりシステムのネットワーク番号が同じであるか)、または別の (リモート) ネットワーク上にあるかによって異なります。

同一ネットワーク上のホスト:

接続のもう一方の終端が同一の IP ネットワーク上に存在する場合、TCP から通知される MSS は、ローカル・ネットワーク・インターフェースの MTU によって決まります。

```
TCP MSS = MTU - TCP header size - IP header size
```

TCP のサイズは 20 バイト、 IPv4 ヘッダー・サイズは 20 バイト、IPv6 ヘッダー・サイズは 40 バイトです。

これは、IP にフラグメント化を起こさずに収容できる考えられる最大の MSS であるため、この値は本質的に最適であり、ローカル・ネットワークの MSS をチューニングする必要がなくなります。

異なるネットワーク上のホスト:

もう一方の接続端がリモート・ネットワーク上にある場合は、このオペレーティング・システムの TCP が、以下の方法で決定される MSS を通知するというデフォルトに設定されます。

その方法は、TCP パス MTU ディスカバリーが使用可能かどうかによって異なります。
tcp_pmtu_discover=0 で、パス MTU ディスカバリーが使用不可になっている場合、 TCP は、以下の順番でどの MSS を使用するかを決定します。

1. **route add** コマンドでこの経路の MTU サイズを指定した場合は、その MTU サイズから MSS が計算されます。
2. 使用するネットワーク・インターフェースのために、ISNO の **tcp_mssdflt** パラメーターを定義した場合は、その **tcp_mssdflt** 値が MSS のために使用されます。

3. 上記のどちらも定義されていない場合、TCP はグローバル **no tcp_mssdflt** チューニング・オプション値を使用します。 このオプションのデフォルト値は 1460 バイトです。

TCP パス **MTU** ディスカバリー**:**

AIX では、TCP パス MTU ディスカバリー・プロトコル・オプションがデフォルトで使用可能です。 このオプションにより、プロトコル・スタックは、2 つのホスト間のパスに存在するネットワーク上の最小 MTU サイズを決定できます。このオプションは **tcp_pmtu_discover=1** ネットワーク・オプションによって制御されます。

TCP パス MTU ディスカバリーの実装には、ICMP ECHO メッセージではなく、接続そのものの TCP パケットが使用されます。TCP/IP カーネル・エクステンションは、PMTU ディスカバリー関連情報を保管するための、PMTU テーブルというテーブルを維持します。宛先への TCP 接続が確立されると、PMTU テーブルにその宛先のエントリーが作成されます。 PMTU 値は、発信インターフェースの MTU 値です。

TCP パケットは、IP ヘッダーの「Don't Fragment」(DF) ビットを設定して送信されます。 MTU 値が TCP パケットのサイズより小さいネットワーク・ルーターに TCP パケットが到達した場合、そのルーターは、フラグメント化できないためにメッセージを転送できないことを示す ICMP エラー・メッセージを送り返します。 エラー・メッセージを送信するルーターがコンパイルに RFC 1191 を使用していれば、ICMP エラー・メッセージにネットワークの MTU 値が含められます。 そうでない場合に TCP パケットを再送するには、AIX TCP/IP カーネル・エクステンション内の既知の MTU 値のテーブルから、もっと小さい値を MTU サイズに割り当てる必要があります。 それから PMTU テーブル内の宛先の PMTU 値が、小さくした新しい MTU サイズに更新され、TCP パケットが再送されます。以降のその宛先への TCP 接続では、更新された PMTU 値が使用されます。

pmtu コマンドを使用して、PMTU エントリーの表示または削除を行えます。 以下に、**pmtu** コマンドの例を示します。

```
# pmtu display

   dst           gw            If    pmtu     refcnt   redisc_t    exp

 --------------------------------------------------------------------------


10.10.1.3    10.10.1.5        en1    1500        2          9       0

10.10.2.5    10.10.2.33       en0    1500        1          0       0
```

使われていない PMTU エントリー (値が 0 の `refcnt` エントリー) は、PMTU テーブルが大きくなるのを防ぐために削除されます。 使われていないエントリーは、`refcnt` 値が 0 になってから *pmtu_expire* 分後に削除されます。*pmtu_expire* ネットワーク・オプションのデフォルト値は 10 分です。 PMTU エントリーが期限切れにならないようにするには、*pmtu_expire* 値を 0 に設定します。

この TCP パス MTU ディスカバリーのインプリメンテーションでは、経路のクローン作成は不要です。このことは、経路指定テーブルが小さくなり管理しやすくなることを意味します。

静的経路**:**

MSS のデフォルト値である 1460 バイトは、特定のリモート・ネットワークに静的経路を指定することによって、変更できます。

route コマンドの **-mtu** オプションを使用して、そのネットワークに MTU を指定してください。 この場合は、MSS 値を計算するのではなく、実際の経路の最小 MTU を指定します。 例えば、以下のコマンドは、ネットワーク 192.3.3 の経路のデフォルト MTU サイズを 1500 に、そのゲートウェイに到達するデフォルト・ホストを en0host2 を設定します。

```
# route add -net 192.1.0 jack -mtu 1500
1500 net 192.3.3: gateway en0host2
```

netstat -r コマンドは、経路テーブルを表示し、 PMTU サイズが 1500 バイトであることを示します。 TCP はその MTU サイズに基づいて MSS を計算します。 以下に、**netstat -r** コマンドの例を示します。

```
# netstat -r
Routing tables
Destination      Gateway           Flags   Refs     Use  If   PMTU Exp Groups

Route tree for Protocol Family 2 (Internet):
default          res101141         UGc       0        0  en4     -   -
ausdns01.srv.ibm res101141         UGHW      8       40  en4  1500   -
10.1.14.0        server1           UHSb      0        0  en4     -   -  =>
10.1.14/24       server1           U         5     4043  en4     -   -
server1          loopback          UGHS      0      125  lo0     -   -
10.1.14.255      server1           UHSb      0        0  en4     -   -
127/8            loopback          U         2  1451769  lo0     -   -
192.1.0.0        en0host1          UHSb      0        0  en0     -   -  =>
192.1.0/24       en0host1          U         4       13  en0     -   -
en0host1         loopback          UGHS      0        2  lo0     -   -
192.1.0.255      en0host1          UHSb      0        0  en0     -   -
192.1.1/24       en0host2          UGc       0        0  en0     -   -
en1host1         en0host2          UGHW      1   143474  en0  1500   -
192.3.3/24       en0host2          UGc       0        0  en0  1500   -
192.6.0/24       en0host2          UGc       0        0  en0     -   -

Route tree for Protocol Family 24 (Internet v6):
loopbackv6       loopbackv6        UH        0        0  lo0 16896   -
```

注**: netstat -r** コマンドは PMTU 値を表示しません。 PMTU 値は **pmtu display** コマンドで表示することができます。 **route add** コマンドで宛先の経路を追加し、MTU 値を指定すると、PMTU テーブルにその宛先の PMTU エントリーが作成されます。

小規模で安定した環境の場合は、この方法によって、ネットワーク単位の MSS の制御が正確になります。この方法の欠点は、次のとおりです。

- 動的経路指定では機能しない。
- リモート・ネットワークの数が増えると、非実用的になる。
- 両端に静的経路を設定して、デフォルトの MSS より大きい MSS を両端にネゴシエーションさせる必要がある。

no コマンドの **tcp_mssdflt** オプションの使用法**:**

tcp_mssdflt オプションは、リモート・ネットワークとの通信に使用される最大パケット・サイズを設定するためのものです。

no コマンドのグローバル *tcp_mssdflt* オプションは、すべてのネットワークに適用されます。 ただし、ISNO オプションをサポートするネットワーク・インターフェースの場合には、それぞれのインターフェースで *tcp_mssdflt* オプションを設定できます。 この値は、ネットワークを使用する経路のグローバル **no** コマンド値をオーバーライドします。

tcp_mssdflt オプションは、TCP データ・サイズを表す TCP MSS サイズです。 この MSS サイズを計算するには、望ましいネットワーク MTU サイズから 40 バイトを差し引きます (IP ヘッダーの 20 と TCP ヘッダーの 20)。 *rfc1323* プロトコル・オプションなどの他のオプションを使用する場合、TCP がこの調整を処理するので、その他のオプションのための調整は必要ありません。

MTU がデフォルトより大きい環境では、この方法に、MSS をネットワーク単位で設定する必要がないという利点があります。 欠点は次のとおりです。

- デフォルトを大きくすると、宛先が本当にリモートであるネットワーク上に存在し、介在するネットワークの MTU が既知でない場合は、IP ルーターでフラグメント化となることがある。
- *tcp_mssdflt* オプションを宛先ホストの値と同じ値に設定する必要がある。

注**:** AIX から、*tcp_pmtu_discover* オプションを 0 に設定したときに使用できるのは *tcp_mssdflt* オプションのみになりました。

サブネット化と **no** コマンドの **subnetsarelocal** オプション**:**

no コマンドの *subnetsarelocal* オプションを使用して、TCP がリモート端点をローカル (同一ネットワーク上) と見なす場合と、リモートと見なす場合とを制御できます。

サブネット化することによって、同じネットワーク番号を共用するには、幾つかの物理ネットワークを作成します。 *subnetsarelocal* オプションは、システム全体にわたって、サブネットがローカル・ネットワークであるかリモート・ネットワークであるかを指定します。 **no -o subnetsarelocal=1** コマンド (デフォルト) を使用すると、サブネット 1 上のホスト A は、サブネット 2 上のホスト B を同じ物理ネットワーク上にあると見なします。

結果として、ホスト A とホスト B が接続を確立するときに、同じネットワーク上にあるものとして MSS をネゴシエーションします。 各ホストは、そのネットワーク・インターフェースの MTU に基づいた MSS を通知するので、通常は最適な MSS が選択されることになります。

この方法の利点は、次のとおりです。

- 静的割り当ての必要がない。MSS は自動的にネゴシエーションされます。
- 隣接するサブネット間の MTU の小さな相違点が適切に処理されるようにするため、TCP MSS のネゴシエーションを使用不可にしたり、変更したりしない。

この方法の欠点は、次のとおりです。

- 2 つの MTU が大きいネットワークが MTU の小さいネットワークを介してリンクされると、IP ルーターのフラグメント化が起きる可能性がある。 次の図は、この問題を示しています。

図 *21.* サブネット間でのフラグメント化： この図に示したデータ・パスは、ホスト A から MTU=4352 の FDDI を経由し、ルーター 1 を経由して、MTU=1500 のイーサネットに至ります。 そこからデータ・パスはルーター 2 に進み、他の MTU=4352 の FDDI に進み、ホスト B に到達します。この例でどのようにフラグメント化が起きるかについては、図のすぐ下の本文で説明します。

- このシナリオでは、ホスト A と B が、共通の 4352 という MTU に基づいて接続を確立する。 A から B へ送られるパケットは、ルーター 1 によって分割され、ルーター 2 によって再組み立てされます。B から A へ送られるときは、これが逆になります。
- 送信元と宛先がどちらも、サブネットをローカルと見なす。

注**:** *tcp_pmtu_discover* 値が 1 の場合は、MSS 値は発信インターフェースの MTU に基づいて計算されます。 *subnetsarelocal* 値が考慮されるのは、*tcp_pmtu_discover* ネットワーク・オプション値が 0 の場合のみです。

IP プロトコルのパフォーマンス・チューニングの推奨事項

このセクションでは、IP プロトコル・パフォーマンスの最適化に関する推奨を提示します。

IP レイヤーで唯一のチューナブル・パラメーターは、*ipqmaxlen* です。これは、IP 入力キューの長さを制御します。 一般に、インターフェースはキューイングを行いません。 パケットは非常に迅速に到達し、IP 入力キューがオーバーランすることがあります。オーバーフロー・カウンター (`ipintrq overflows`) を表示するには、**netstat -s** または **netstat -p ip** コマンドを使用します。

戻された数が 0 より大きい場合は、オーバーフローが起きています。 このキューの最大長を設定するには、**no** コマンドを使用してください。次に例を示します。

```
# no -o ipqmaxlen=100
```

この例では、100 個のパケットをキューに入れることができます。 使用する正確な値は、受け取った最大バースト率によって判別できます。 これが判別できない場合は、オーバーフロー数を使用すると、何が増加しているかを判別するのに役立ちます。 キューの長さを長くしても、追加のメモリーは使用されません。 ただし、キューの長さを長くすると、IP の入力キュー上に処理すべきパケットが増えるので、結果的にオフ・レベルの割り込みハンドラーで経過する時間が増えることがあります。 さらに、これによって、CPU 時間を必要とするプロセスが影響を受ける可能性があります。 パケット・ドロップの低減とその他の処理に必要な CPU 可用性のトレードオフになります。 現在の環境でトレードオフが問題となる場合は、*ipqmaxlen* を適度な増分で大きくするのが最善策です。

mbuf プールのパフォーマンスのチューニング

ネットワーク・サブシステムでは、mbuf と呼ばれるデータ構造体を中心としたメモリー管理機能を使用します。

mbuf は多くの場合、カーネル内の着信および送信ネットワーク・トラフィックのデータを保管するために使用されます。 mbuf プールのサイズを適切なサイズにすることは、ネットワーク・パフォーマンスに良い影響を与えます。 mbuf プールの構成が誤っていると、ネットワークとシステムの両方のパフォーマンスが低下することがあります。 mbuf プール・サイズの上限値である *thewall* チューニング・オプションは、システムのメモリー量に基づいてオペレーティング・システムによって自動的に決定されます。 mbuf プール・サイズの上限をチューニングできるのは、システム管理者だけです。

thewall チューニング・オプション

ネットワーク・チューニング・オプションである *thewall* は、ネットワーク・カーネル・バッファーの上限値を設定します。

thewall チューニング・オプションの値は、システムによって自動的に最大値に設定されます。この値は、基本的に変更するべきではありません。 システムがネットワーク・バッファーのために使用するメモリー量を減らすために、この値を小さくすることは可能ですが、ネットワークのパフォーマンスに影響を与える

場合があります。 システムはその時々で必要な数のバッファーのみを使用するため、ネットワーク・サブシステムが高負荷で使用されていない状況では、合計バッファー数は、*thewall* 値よりもかなり低くなります。

thewall チューニング・オプションの単位は 1 KB なので、1048576 バイトは RAM が 1024 MB、つまり 1GB であることを表します。

mbuf リソース制限

AIX 6.1 には最大 65 GB の mbuf バッファー・スペースがあり、そのスペースはそれぞれ 256 MB の 260 個のメモリー・セグメントで構成されています。

thewall チューニング・オプションの値は、65 GB または、システムのメモリー量の半分のいずれか小さいほうの値となります。

maxmbuf チューニング・オプション

maxmbuf チューニング・オプションの値は、通信サブシステムが使用する実メモリーの量を制限します。

maxmbuf チューニング・オプションを使用して *thewall* の制限を下げることもできます。 *maxmbuf* チューニング・オプションの値を表示するには、**lsattr -E -l sys0** コマンドを実行します。 *maxmbuf* の値が 0 より大きい場合は、 *thewall* の値に関係なく *maxmbuf* 値が使用されます。

maxmbuf チューニング・オプションのデフォルト値は 0 です。 *maxmbuf* チューニング・オプションの値が 0 であることは、*thewall* チューニング・オプションが使用されることを意味します。 *maxmbuf* チューニング・オプションの値を変更するには、**chdev** または **smitty** コマンドを使用します。

sockthresh および strthresh しきい値チューニング・オプション

sockthresh および *strthresh* チューニング・オプションは、新しいソケットや TCP 接続のオープン、または新しいストリーム・リソースの作成を制限するための上限しきい値です。 これによってバッファー・リソースが使用不可になることを防ぎ、既存のセッションや接続が処理を継続するためのリソースを必ず持てるようにします。

sockthresh チューニング・オプションは、メモリー使用量を制限します。 新しいソケット接続が、*sockthresh* チューニング・オプションの値を超えることはできません。 *sockthresh* チューニング・オプションのデフォルト値は 85% で、割り当てられた合計メモリー量が *thewall* または *maxmbuf* チューニング・オプションの値の 85% に達した場合、これ以上新しいソケット接続を持つことはできません。これは、バッファーの使用量が 85% 未満になるまで、**socket()** および **socketpair()** システム・コールの戻り値が *ENOBUFS* になることを意味します。

同様に、*strthresh* チューニング・オプションは、ストリーム・リソースのための mbuf メモリーの使用量を制限します。*strthresh* チューニング・オプションのデフォルト値は、85% です。 async および TTY サブシステムは、ストリーム環境で実行されます。 *strthresh* チューニング・オプションは、合計割り当てメモリー量が *thewall* チューニング・オプションの 85% に達した場合に、それ以上のメモリーがストリーム・リソースに行かないように指定します。これは、ストリームのオープン、モジュールのプッシュ、またはストリーム・デバイスへの書き込みのためのストリーム・コールの戻り値が *ENOSR* であることを意味します。

sockthresh および *strthresh* しきい値は、**no** コマンドを使用してチューニングできます。

mbuf 管理機能

mbuf 管理機能によって、32 バイトから 16384 バイトまでの各種バッファー・サイズが制御されます。

プールは、仮想メモリー・マネージャー (VMM) に割り当て要求を出すことによって、システム・メモリーから作成されます。 プールは、カーネル仮想メモリーの固定部分から構成されています。カーネル仮想メモリー内で、プールは常に物理メモリー内にあり、ページアウトされることはありません。 結果的に、アプリケーション・プログラムとデータ内のページングに使用できる実メモリーが、mbuf プールが増加した分だけ減少します。

ネットワーク・メモリー・プールは、各プロセッサーに平等に分けられます。 各サブプールは複数のバケットに分割され、それぞれのバケットは、32 から 16384 バイトの複数のバッファーを保持します。 各バケットは、同じプロセッサー上のほかのバケットからメモリーを借りることができますが、プロセッサーは別のプロセッサーのネットワーク・メモリー・プールからメモリーを借りることはできません。 ネットワーク・サービスがデータを転送する必要があるときには、**m_get()** などのカーネル・サービスを呼び出して、メモリー・バッファーを取得します。 バッファーが既に使用可能であり、固定されている場合は、即時に取得できます。 上限に達しておらず、かつ、バッファーが固定されていない場合は、バッファーが割り当てられ、固定されます。 一度固定されると、メモリーは固定されたままになりますが、ネットワーク・プールに解放することはできます。 フリー・バッファーの数が上限に達すると、特定数の固定が解除され、一般的に使用できるようにシステムに戻されます。 この固定解除は、**netm()** カーネル・プロセスによって行われます。 **m_get()** サブルーチンの呼び出し元は、ネットワーク・メモリー・バッファーを待機するかどうかを指定できます。 **M_DONTWAIT** フラグが指定され、その時点で使用可能な固定バッファーがない場合は、「failed」カウンターが増分されます。 **M_WAIT** フラグが指定された場合は、バッファーを割り当てて固定できるようになるまで、プロセスがスリープ状態になります。

mbuf プールをモニターするための netstat -m コマンド

ネットワーク・メモリー要求 (mbufs/clusters) の不足や失敗を検出するには、**netstat -m** コマンドを使用します。

mbuf 統計情報をクリアする (ゼロにする) には、**netstat -Zm** コマンドを使用します。 これは、テストの実行を開始する際に統計情報をクリアする場合に役立ちます。 以下のフィールドが、**netstat -m** コマンドに用意されています。

フィールド名
: 定義

By size
: バッファー・サイズを表示します。

inuse
: 使用中の特定サイズのバッファー数を表示します。

calls
: バッファー・サイズごとの呼び出し回数、または割り当て要求の回数を表示します。

failed
: 使用可能バッファーがないことによる割り当て要求の失敗回数を表示します。

delayed
: 特定サイズのバッファーが空で、呼び出し元が **M_WAIT** フラグを設定したことによって呼び出しが遅延した回数を表示します。

free
: フリー・リストに存在し、割り当て可能な状態になっている各サイズのバッファー数を表示します。

hiwat
: フリー・リストに残される最大バッファー数 (システムによって決定される値) を表示します。 この上限を超えたフリー・バッファーは、システムに対してゆっくりと解放されていきます。

freed
: フリー・カウントが **hiwat** 制限を超えたためにシステムに対して解放されたバッファーの数を表示します。

failed 呼び出しの数が大きくなるようであってはなりません。 少数でも存在すれば、そのことがトリガーになって、バッファー・プール・サイズの増加に伴い、システムがさらにバッファーを割り当てることになります。 リブート後のシステム開始時には、各サイズのバッファーの事前定義セットが用意されますが、その後、必要に応じてバッファー数が増加します。

2 つのプロセッサーまたは CPU を搭載したマシンの **netstat -m** コマンドの例を以下に示します。

```
# netstat -m

Kernel malloc statistics:

******* CPU 0 *******
By size           inuse     calls failed   delayed    free   hiwat   freed
32                   68       693      0         0      60    2320       0
64                   55       115      0         0       9    1160       0
128                  21       451      0         0      11     580       0
256                1064      5331      0         0    1384    1392      42
512                  41       136      0         0       7     145       0
1024                 10       231      0         0       6     362       0
2048               2049      4097      0         0     361     362     844
4096                  2         8      0         0     435     435     453
8192                  2         4      0         0       0      36       0
16384                 0       513      0         0      86      87     470


******* CPU 1 *******
By size           inuse     calls failed   delayed    free   hiwat   freed
32                 139       710      0         0     117    2320       0
64                  53       125      0         0      11    1160       0
128                 41       946      0         0      23     580       0
256                 62      7703      0         0    1378    1392     120
512                 37       109      0         0      11     145       0
1024                21       217      0         0       3     362       0
2048                 2      2052      0         0     362     362     843
4096                 7        10      0         0     434     435     449
8192                 0         4      0         0       1      36       0
16384                0      5023      0         0      87      87    2667


***** Allocations greater than 16384 Bytes *****

By size           inuse     calls failed   delayed    free   hiwat   freed
65536                2         2      0         0       0    4096       0

Streams mblk statistic failures:
0 high priority mblk failures
0 medium priority mblk failures
0 low priority mblk failures
```

ARP キャッシュのチューニング

アドレス解決プロトコル (ARP) は、32 ビット IPv4 アドレスをデータ・リンク・プロトコルで必要とされる 48 ビット・ホスト・アダプター・アドレスにマップする場合に使用されます。

ARP は、システムによって透過的に処理されます。 しかし、システムは 32 ビット IP アドレスと 48 ビット・ホスト・アドレスの関連を保持するテーブルである ARP キャッシュを保守します。 多数のマシン (クライアント) を接続する環境では、ARP キャッシュのサイズを変更しなければならない場合があります。この場合は、**no** および **netstat** コマンドを使用して行います。

no コマンドはネットワークのチューニング・パラメーターを構成します。そして、この ARP 関連チューナブル・パラメーターは、以下のとおりです。

- **arpqsize = 12**

- **arpt_killc = 20**
- **arptab_bsiz = 7**
- **arptab_nb = 149**

ARP テーブル・サイズは、*arptab_nb* パラメーターで定義されている数のバケットで構成されています。各バケットは **arptab_bsiz** パラメーターに定義済みのエントリー数を保持します。デフォルトは、149 個のバケットであり、それぞれに 7 個のエントリーがあります。したがって、このテーブルは、1043 (149 x 7) 個のホスト・アドレスを保持できます。 このデフォルト設定は IP ネットワーク上で 1043 個までの他のマシンと同時に通信することができるシステムで機能します。 1 つのサーバーがネットワーク上のマシンと 1043 を超えて同時に接続されると、ARP テーブルが小さすぎて ARP テーブルがスラッシングを起こし、パフォーマンスが低下します。 オペレーティング・システムはキャッシュ内のエントリーをパージし、新しいアドレスで置き換える必要があります。 TCP または UDP のパケットは、ARP プロトコルがこの情報を交換するまでキューに入る必要があります。 *arpqsize* パラメーターでは、 ARP 要求から ARP 応答が戻されるまでに、いくつの待ちパケットを ARP 層でキューに入れるかを指定します。 ARP キューがオーバーランする場合、発信 TCP または UDP パケットは破棄されます。

ARP キャッシュのスラッシングは、以下の理由によりパフォーマンスに悪影響を与える場合があります。

1. 現在の発信パケットは、ネットワーク上の ARP プロトコルのルックアップを待つ必要があります。
2. その他の ARP エントリーは、ARP キャッシュから除去されなければなりません。 すべてのアドレスが必要な場合、削除されるホスト・アドレスが送信済みパケットを持っていれば、別のアドレスが必要になります。
3. ARP 出力キューがオーバーランし、パケットの破棄を引き起こす可能性があります。

arpqsize、*arptab_bsiz*、および *arptab_nb* パラメーターはいずれもリブート・パラメーターです。つまり、これらの値を変更した場合はブート時または TCP/IP ロード時に構築されるテーブルを変更するために、システムのリブートが必要になります。

arpt_killc パラメーターは、ARP エントリーが削除される前の分単位の時間です。 *arpt_killc* パラメーターのデフォルト値は 20 分です。 ARP エントリーは、ネットワーク・アダプターを取り替えるときにホスト・システム自体が 48 ビット・アドレスを変更することがあるケースをカバーするために、**arpt_killc** パラメーターで定義された時間 (分単位) ごとにテーブルから削除されます。 これにより、キャッシュ内の失効したエントリーは必ず削除され、古いアドレスが削除されるまでの間このようなホストとの通信が不可になります。 この時間を長く設定すれば、システムによる ARP のルックアップの数を削減できますが、失効したホスト・アドレスの保持時間がそれだけ長くなります。 *arpt_killc* パラメーターは動的パラメーターなので、システムのリブートなしで変更することができます。

netstat -p arp コマンドは、ARP 統計情報を表示します。これらの統計情報は、合計でいくつの ARP 要求が送信されたか、および新しいエントリーの領域を作るためにエントリーを削除するときに、いくつのパケットがテーブルからパージされたかを表示します。 このパージ・カウントが高い場合は、ARP テーブル・サイズを増やす必要があります。 以下に、**netstat -p arp** コマンドの例を示します。

```
# netstat -p arp

arp:
     6 packets sent
     0 packets purged
```

arp -a コマンドを使用して、ARP テーブルを表示できます。このコマンドの出力は、どのアドレスが ARP テーブルに存在するか、これらのアドレスがどのようにハッシュされているか、どのバケットにハッシュされているかを表示します。

```
 ? (10.3.6.1) at 0:6:29:dc:28:71 [ethernet] stored

bucket:    0     contains:    0 entries
bucket:    1     contains:    0 entries
bucket:    2     contains:    0 entries
bucket:    3     contains:    0 entries
bucket:    4     contains:    0 entries
bucket:    5     contains:    0 entries
bucket:    6     contains:    0 entries
bucket:    7     contains:    0 entries
bucket:    8     contains:    0 entries
bucket:    9     contains:    0 entries
bucket:   10     contains:    0 entries
bucket:   11     contains:    0 entries
bucket:   12     contains:    0 entries
bucket:   13     contains:    0 entries
bucket:   14     contains:    1 entries
bucket:   15     contains:    0 entries


...lines omitted...

There are 1 entries in the arp table.
```

ネーム・レゾリューションのチューニング

TCP/IP は、ネーム・レゾリューション と呼ばれるプロセスの中で、ホスト名から IP アドレスを取得しようとします。

IP アドレスをホスト名に変換するプロセスは、逆ネーム・レゾリューション と呼ばれます。 名前を解決するには、*resolver* ルーチン が使用されます。 このルーチンは、DNS、NIS、および最終的にはローカルの /etc/hosts ファイルを照会して、必要な情報を検索します。

名前をどのように解決するか分かっている場合は、デフォルトの検索順序を上書きすることによって、ネーム・レゾリューションのプロセスの速度を上げることができます。 これは、/etc/netsvc.conf ファイルまたは *NSORDER* 環境変数を使用して行います。

- /etc/netsvc.conf ファイルと *NSORDER* の両方が使用されると、*NSORDER* によって /etc/netsvc.conf ファイルが変更されます。 /etc/netsvc.conf を使用してホストの順序付けを指定するには、このファイルを作成し、次の行を入力します。

  ```
  hosts=value,value,value
  ```

  この *value* (小文字のみ) は bind、local、nis、bind4、bind6、local4、local6、nis4、または nis6 (/etc/hosts の場合) のいずれかです。 順序は 1 行中に指定し、各値をコンマで区切ります。 コンマと等号の間にはホワイト・スペースを入れても構いません。

  指定される値とその順序付けは、ネットワーク構成によって異なります。 例えば、ローカル・ネットワークがフラット・ネットワークとして編成されている場合は、/etc/hosts ファイルだけが必要です。 /etc/netsvc.conf ファイルには次の行を入力します。

  ```
  hosts=local
  ```

  NSORDER 環境変数は次のように設定します。

  ```
  NSORDER=local
  ```

- ローカル・ネットワークが、ネーム・レゾリューションにネームサーバーを使用し、バックアップに /etc/hosts ファイルを使用するドメイン・ネットワークである場合は、両方のサービスを指定します。 /etc/netsvc.conf ファイルには次の行を入力します。

```
hosts=bind,local
```

NSORDER 環境変数は次のように設定します。

```
NSORDER=bind,local
```

このアルゴリズムでは、リスト内の最初のソースを試行します。 次に、アルゴリズムは、以下に基づいて、別の指定されたサービスを試行するかどうか判断します。

- 現在のサービスが実行中でないため、使用できない。
- 現在のサービスが、名前を見つけることができなかったため、権限がない。

ネットワーク・パフォーマンスの分析

パフォーマンス上の問題が起きた場合は、システムにはまったく原因はなく、実際の原因は消え去っていることがあります。

ネットワークがパフォーマンス全体に影響を与えるかどうかを見分ける簡単な方法は、ネットワークが関係する操作と、関係しない操作とを比較することです。大量のリモート読み取りと書き込みを行うプログラムを実行し、その実行が低速である場合、それ以外のものがすべて正常に実行されているときには、ネットワーク上の問題が原因であると考えられます。 一部の潜在的なネットワークのボトルネックは、次のことが原因で生じることがあります。

- クライアント・ネットワークのインターフェース
- ネットワークの帯域幅
- ネットワーク・トポロジー
- サーバー・ネットワークのインターフェース
- サーバーの CPU ロード
- サーバーのメモリー使用率
- サーバーの帯域幅
- 非効率な構成

ネットワーク統計情報を測るツールが幾つかあり、各種の情報が提供されますが、パフォーマンス・チューニングに関連するのはその情報の一部だけです。

パフォーマンスを改善するには、**no** (ネットワーク・オプション) コマンドと、NFS オプションをチューニングするための **nfso** コマンドを使用します。また、**chdev** および **ifconfig** コマンドを使用して、システムとネットワーク・パラメーターを変更することもできます。

ping コマンド

ping コマンドはネットワークと各種外部ホスト状況の判別、ハードウェアとソフトウェア問題のトラッキングと切り分け、およびネットワークのテスト、測定、管理に役立ちます。

パフォーマンス・チューニングと関係のある **ping** コマンド・オプションは、次のとおりです。

-c パケットの数を指定します。 このオプションは、IP トレース・ログを取得するときに役立ちます。 最小数の **ping** パケットを取り込むことができます。

-s パケットの長さを指定します。 このオプションを使用して、フラグメント化と再組み立てを調べることができます。

-f パケットを 10 ミリ秒間隔で、または各応答後に即時に送信します。このオプションを使用できるのは、root ユーザーだけです。

ネットワークまたはシステムをロードする必要がある場合は、**-f** オプションが便利です。例えば、問題の原因が過負荷であると考えられる場合は、それを確認するため、故意に環境をロードします。 幾つかの **aixterm** ウィンドウを開き、各ウィンドウで **ping -f** コマンドを実行します。 イーサネット使用率はすぐに 100% 近くになります。 次に例を示します。

```
# date; ping -c 1000 -f 192.1.6.1 ; date
Thu Feb 12 10:51:00 CST 2004
PING 192.1.6.1 (192.1.6.1): 56 data bytes
.
--- 192.1.6.1 ping statistics ---
1000 packets transmitted, 1000 packets received, 0% packet loss
round-trip min/avg/max = 1/1/23 ms
Thu Feb 12 10:51:00 CST 2004
```

注**: ping** コマンドをネットワーク上で使用するのは大変難しい場合があるので、使用するときには十分な注意が必要です。 フラッド ping を実行できるのは、root ユーザーだけです。

この例では、1 秒間に 1000 個のパケットが送信されています。 このコマンドでは、IP と ICMP (Internet Control Message Protocol) プロトコルを使用しているので、転送プロトコル (UDP/TCP) とアプリケーションのアクティビティーが関係していないことに注目してください。 測定されたデータ (例えば、往復時間など) は、全体のパフォーマンス特性を反映しません。

大量のパケットを宛先に送信しようとするときには、次の幾つかの点を考慮してください。

- 送信パケットは、システムにロードを書き込む。
- 測定中にネットワーク・インターフェースの状況をモニターするには、**netstat -i** コマンドを使用する。 「Oerrs」の出力を調べると、送信中にシステムがパケットを廃棄していることを確認できる場合があります。
- mbuf および送受信キューなどの、その他のリソースもモニターする必要がある。 宛先システムが過負荷になることはあまりありません。ほかのシステムが過負荷になる前に、使用中のシステムが過負荷になることがあります。
- 結果の関連性を考える。 1 つの宛先システムだけをモニターまたはテストする場合は、ネットワークまたはルーターに問題がある可能性を考えて、比較のためにほかのシステムにも同じことを行う必要があります。

ftp コマンド

ftp コマンドを使用すると、入力として **/dev/zero** を使用し、出力として **/dev/null** を使用することによって、非常に大きいファイルを送信することができます。 これにより、ディスクを利用せずに (ボトルネックとなる可能性があるため)、またメモリー内のファイル全体をキャッシュに入れる必要なく、ラージ・ファイルを転送できます。

次のように **ftp** サブコマンドを使用してください (**dd** コマンドが読み取るブロック数を増やすか減らすには、count を変更してください)。

```
> bin
> put "|dd if=/dev/zero bs=32k count=10000" /dev/null
```

このコマンドは 10000 ブロックのデータを転送し、各ブロックのサイズは 32 KB です。 転送されるファイルのサイズを大きくまたは小さくするには、**dd** コマンドが読み取るブロックの数 (count パラメーター) を変更するか、ブロック・サイズ (bs パラメーター) を変更します。 **ftp** コマンドでのデフォルトのファイル・タイプは ASCII です。このタイプでは、すべてのバイトがスキャンされる必要があるため、遅くなるので注意してください。 可能であれば、転送には常にバイナリー・モード (bin) を使用するようにしてください。

ギガビット・イーサネットの「ジャンボ・フレーム」の場合、および MTU が 9180 以上の ATM の場合は、MTU サイズが大きいため、高いパフォーマンスを得るには、**tcp_sendspace** と **tcp_recvspace** が 65535 以上でなければなりません。 最適パフォーマンスを得るには、サイズを 131072 バイト (128 KB) にすることをお勧めします。 ギガビット・イーサネット・アダプターを **SMIT** ツールを使用して構成すれば、ISNO システム・デフォルト値は正しく設定されます。 **ifconfig** コマンドを使用してネットワーク・インターフェースを構成すると、ISNO オプションは正しく設定されません。

以下は、パラメーターの設定例です。

```
# no -o tcp_sendspace=65535
# no -o tcp_recvspace=65535
```

ftp サブコマンドは次のようになります。

```
ftp> bin
200 Type set to I.
ftp> put "|dd if=/dev/zero bs=32k count=10000" /dev/null
200 PORT command successful.
150 Opening data connection for /dev/null.
10000+0 records in
10000+0 records out
226 Transfer complete.
327680000 bytes sent in 2.789 seconds (1.147e+05 Kbytes/s)
local: |dd if=/dev/zero bs=32k count=10000 remote: /dev/null
ftp> quit
221 Goodbye.
```

このデータ転送は、1500 バイトの MTU で 2 つのギガビット・イーサネット・アダプター間で実行されたもので、ここでは 114700 KB/秒というスループットが報告されています。これは 112 MB/秒または 940 Mbps に相当します。

送信側と受信側が MTU サイズを 9000 にしてジャンボ・フレームを使用したときに報告されたスループットは、次の例でお分かりのように 120700 KB/秒 (117.87 MB/秒または 989 Mbps) でした。

```
ftp> bin
200 Type set to I.
ftp> put "|dd if=/dev/zero bs=32k count=10000" /dev/null
200 PORT command successful.
150 Opening data connection for /dev/null.
10000+0 records in
10000+0 records out
226 Transfer complete.
327680000 bytes sent in 2.652 seconds (1.207e+05 Kbytes/s)
local: |dd if=/dev/zero bs=32k count=10000 remote: /dev/null
```

以下は、2 つの 10/100 Mbps イーサネット・インターフェース間の **ftp** データ転送の例です。

```
ftp> bin
200 Type set to I.
ftp> put "|dd if=/dev/zero bs=32k count=10000" /dev/null
200 PORT command successful.
150 Opening data connection for /dev/null.
10000+0 records in
10000+0 records out
226 Transfer complete.
327680000 bytes sent in 27.65 seconds (1.157e+04 Kbytes/s)
local: |dd if=/dev/zero bs=32k count=10000 remote: /dev/null
```

このデータ転送のスループットは 11570 KB/秒で、これは 11.3 MB/秒または 94.7 Mbps に相当します。

netstat コマンド

netstat コマンドは、ネットワーク状況を示すために使用されます。

このコマンドは一般的に、パフォーマンス測定ではなく、問題判別のために使用されるものです。 しかし、**netstat** コマンドは、ネットワーク上のトラフィックの量を判別し、パフォーマンス上の問題がネットワーク輻輳 (ふくそう) によるものかどうかを特定するために使用することもできます。

netstat コマンドでは、構成されたネットワーク・インターフェース上のトラフィックに関連する次のような情報を示します。

- ソケットに関連付けられたプロトコル制御ブロックのアドレスと、すべてのソケットの状態
- 通信サブシステムで受信、送信、および廃棄されたパケットの数
- インターフェースごとの累積統計情報
- 経路とその状況

netstat コマンドの使用法**:**

netstat コマンドは、アクティブ接続の各種ネットワーク関連データ構造体の内容を示します。

netstat -in コマンド**:**

netstat 関数は構成済みのすべてのインターフェースの状態を表示します。

次の例は、内蔵イーサネット (en1)、PCI-X ギガビット・イーサネット (en0)、および TCP/IP 用に構成されたファイバー・チャネル・アダプター (fc0) が組み込まれたワークステーションの統計情報です。

```
# netstat -in
Name  Mtu   Network     Address              Ipkts Ierrs    Opkts Oerrs  Coll
en1   1500  link#2      0.9.6b.3e.0.55       28800     0      506     0     0
en1   1500  10.3.104    10.3.104.116         28800     0      506     0     0
fc0   65280 link#3      0.0.c9.33.17.46         12     0       11     0     0
fc0   65280 192.6.0.0   192.6.0.1               12     0       11     0     0
en0   1500  link#4      0.2.55.6a.a5.dc         14     0       20     5     0
en0   1500  192.1.6     192.1.6.1               14     0       20     5     0
lo0   16896 link#1                           33339     0    33343     0     0
lo0   16896 127         127.0.0.1            33339     0    33343     0     0
```

システム起動以降のカウント値が合計されます。

Name インターフェース名。

Mtu 最大伝送単位。 インターフェースを使用して送信されるパケットの最大サイズ (バイト単位)。

Ipkts 受信されたパケットの合計数。

Ierrs 入力エラーの合計数。 例えば、無効な形式のパケット、チェックサム・エラー、またはデバイス・ドライバーでのバッファー・スペース不足など。

Opkts
送信されたパケットの合計数。

Oerrs 出力エラーの合計数。 例えば、ローカル・ホスト接続での障害、またはアダプター出力キューのオーバーランなど。

Coll 検出されたパケット衝突の数。

注**: netstat -i** コマンドは、イーサネット・インターフェースの衝突カウントをサポートしません (イーサネットの統計情報については、 353 ページの『アダプター統計情報』を参照してください)。

以下は、チューニングに関するガイドラインです。

- 入力パケット中のエラーの数 (**netstat -i** コマンドで検出) が、次のように入力パケット合計数の 1% を超える。

  ```
  Ierrs > 0.01 x Ipkts
  ```

  この場合は、**netstat -m** コマンドを実行して、メモリー不足について調べてください。
- 出力パケット中のエラーの数 (**netstat -i** コマンドで検出) が、次のように出力パケット合計数の 1% を超える。

  ```
  Oerrs > 0.01 x Opkts
  ```

  この場合は、そのインターフェースの送信キュー・サイズ (*xmt_que_size*) を大きくしてください。*xmt_que_size* のサイズは、次のコマンドで調べることができます。

  ```
  # lsattr -El adapter
  ```
- 衝突率が次のように 10% を超える。

  ```
  Coll / Opkts > 0.1
  ```

  この場合は、ネットワーク使用率が高いので、再編成または分割化が必要な場合があります。 **netstat -v** または **entstat** コマンドを使用して、衝突率を判別してください。

netstat -i -Z コマンド*:*

netstat コマンドのこの機能では、 **netstat -i** コマンドの統計カウンターをすべてクリアして、ゼロにします。

netstat -I インターフェース間隔*:*

この **netstat** 関数は指定インターフェースの統計情報を表示します。

この関数は指定インターフェースに **netstat -i** コマンドと同様の情報を提供し、指定時間間隔でこの情報を報告します。 次に例を示します。

```
# netstat -I en0 1
    input   (en0)      output           input   (Total)    output
   packets  errs  packets  errs colls  packets  errs  packets   errs colls
       0     0       27     0     0   799655     0   390669     0     0
       0     0        0     0     0        2     0        0     0     0
       0     0        0     0     0        1     0        0     0     0
       0     0        0     0     0       78     0      254     0     0
       0     0        0     0     0      200     0       62     0     0
       0     0        1     0     0        0     0        2     0     0
```

この例では、ent0 インターフェースに関する **netstat -I** コマンドの出力を示しています。 2 つのレポートが並行して生成されます。1 つは指定されたインターフェースのもので、もう 1 つは使用可能なすべてのインターフェースのもの (「Total」) です。 フィールドは、**netstat -i** の例と同様で、「input packets」=「Ipkts」、「input errs」=「Ierrs」などがあります。

netstat -a コマンド*:*

netstat -a コマンドは、すべてのソケットの状態を表示します。

-a フラグを指定しない場合は、サーバー・プロセスで使用されているソケットは表示されません。次に例を示します。

```
# netstat -a
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp4       0      0  *.daytime              *.*                    LISTEN
tcp        0      0  *.ftp                  *.*                    LISTEN
tcp        0      0  *.telnet               *.*                    LISTEN
tcp4       0      0  *.time                 *.*                    LISTEN
tcp4       0      0  *.sunrpc               *.*                    LISTEN
tcp        0      0  *.exec                 *.*                    LISTEN
tcp        0      0  *.login                *.*                    LISTEN
tcp        0      0  *.shell                *.*                    LISTEN
tcp4       0      0  *.klogin               *.*                    LISTEN
tcp4       0      0  *.kshell               *.*                    LISTEN
tcp        0      0  *.netop                *.*                    LISTEN
tcp        0      0  *.netop64              *.*                    LISTEN
tcp4       0   1028  brown10.telnet         remote_client.mt.1254  ESTABLISHED
tcp4       0      0  *.wsmserve             *.*                    LISTEN
udp4       0      0  *.daytime              *.*
udp4       0      0  *.time                 *.*
udp4       0      0  *.sunrpc               *.*
udp4       0      0  *.ntalk                *.*
udp4       0      0  *.32780                *.*
Active UNIX domain sockets
SADR/PCB  Type   Recv-Q Send-Q Inode    Conn     Refs     Nextref  Addr
71759200 dgram       0      0 13434d00        0        0        0 /dev/SRC
7051d580
71518a00 dgram       0      0 183c3b80        0        0        0 /dev/.SRC-unix/SRCCwfCEb
```

netstat -ao コマンドを使用して、各ソケットの詳細情報を表示することができます。次の例では、**ftp** ソケットはジャンボ・フレーム用に構成されたギガビット・イーサネット・アダプター上で実行されています。

```
# netstat -ao

Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
[...]

tcp4       0      0  server1.ftp           client1.33122         ESTABLISHED

         so_options: (REUSEADDR|OOBINLINE)
         so_state: (ISCONNECTED|PRIV)
         timeo:0 uid:0
         so_special: (LOCKBALE|MEMCOMPRESS|DISABLE)
         so_special2: (PROC)
         sndbuf:
                 hiwat:134220 lowat:33555 mbcnt:0 mbmax:536880
         rcvbuf:
                 hiwat:134220 lowat:1 mbcnt:0 mbmax:536880
                 sb_flags: (WAIT)
         TCP:
         mss:8948  flags: (NODELAY|RFC1323|SENT_WS|RCVD_WS|SENT_TS|RCVD_TS)


tcp4       0      0  server1.telnet         sig-9-49-151-26..2387  ESTABLISHED

         so_options: (REUSEADDR|KEEPALIVE|OOBINLINE)
         so_state: (ISCONNECTED|NBIO)
         timeo:0 uid:0
         so_special: (NOUAREA|LOCKBALE|EXTPRIV|MEMCOMPRESS|DISABLE)
         so_special2: (PROC)
         sndbuf:
                 hiwat:16384 lowat:4125 mbcnt:0 mbmax:65536
                 sb_flags: (SEL|NOINTR)
         rcvbuf:
                 hiwat:66000 lowat:1 mbcnt:0 mbmax:264000
```

```
                sb_flags: (SEL|NOINTR)
        TCP:
        mss:1375

tcp4       0    925  en6host1.login         en6host2.1023          ESTABLISHED

        so_options: (REUSEADDR|KEEPALIVE|OOBINLINE)
        so_state: (ISCONNECTED|NBIO)
        timeo:0 uid:0
        so_special: (NOUAREA|LOCKBALE|EXTPRIV|MEMCOMPRESS|DISABLE)
        so_special2: (PROC)
        sndbuf:
                hiwat:16384 lowat:16384 mbcnt:3216 mbmax:65536
                sb_flags: (SEL|NOINTR)
        rcvbuf:
                hiwat:130320 lowat:1 mbcnt:0 mbmax:521280
                sb_flags: (SEL|NOINTR)
        TCP:
        mss:1448  flags: (RFC1323|SENT_WS|RCVD_WS|SENT_TS|RCVD_TS)



tcp        0      0  *.login                *.*                    LISTEN

        so_options: (ACCEPTCONN|REUSEADDR)
        q0len:0 qlen:0 qlimit:1000      so_state: (PRIV)
        timeo:0 uid:0
        so_special: (LOCKBALE|MEMCOMPRESS|DISABLE)
        so_special2: (PROC)
        sndbuf:
                hiwat:16384 lowat:4096 mbcnt:0 mbmax:65536
        rcvbuf:
                hiwat:16384 lowat:1 mbcnt:0 mbmax:65536
                sb_flags: (SEL)
        TCP:
        mss:512

tcp        0      0  *.shell                *.*                    LISTEN

        so_options: (ACCEPTCONN|REUSEADDR)
        q0len:0 qlen:0 qlimit:1000      so_state: (PRIV)
        timeo:0 uid:0
        so_special: (LOCKBALE|MEMCOMPRESS|DISABLE)
        so_special2: (PROC)
        sndbuf:
                hiwat:16384 lowat:4096 mbcnt:0 mbmax:65536
        rcvbuf:
                hiwat:16384 lowat:1 mbcnt:0 mbmax:65536
                sb_flags: (SEL)
        TCP:
        mss:512


tcp4       0   6394  brown10.telnet         remote_client.mt.1254  ESTABLISHED

        so_options: (REUSEADDR|KEEPALIVE|OOBINLINE)
        so_state: (ISCONNECTED|NBIO)
        timeo:0 uid:0
        so_special: (NOUAREA|LOCKBALE|EXTPRIV|MEMCOMPRESS|DISABLE)
        so_special2: (PROC)
        sndbuf:
                hiwat:16384 lowat:4125 mbcnt:65700 mbmax:65536
                sb_flags: (SEL|NOINTR)
        rcvbuf:
                hiwat:16500 lowat:1 mbcnt:0 mbmax:66000
                sb_flags: (SEL|NOINTR)
        TCP:
```

```
         mss:1375


udp4       0      0  *.time                 *.*

         so_options: (REUSEADDR)
         so_state: (PRIV)
         timeo:0 uid:0
         so_special: (LOCKBALE|DISABLE)
         so_special2: (PROC)
         sndbuf:
                 hiwat:9216 lowat:4096 mbcnt:0 mbmax:36864
         rcvbuf:
                 hiwat:42080 lowat:1 mbcnt:0 mbmax:168320
                 sb_flags: (SEL)

[...]

Active UNIX domain sockets
SADR/PCB  Type   Recv-Q Send-Q Inode    Conn     Refs     Nextref  Addr
71759200 dgram       0      0 13434d00        0        0        0 /dev/SRC
7051d580
         so_state: (PRIV)
         timeo:0 uid:0
         so_special: (LOCKBALE)
         so_special2: (PROC)
         sndbuf:
                 hiwat:8192 lowat:4096 mbcnt:0 mbmax:32768
         rcvbuf:
                 hiwat:45000 lowat:1 mbcnt:0 mbmax:180000
                 sb_flags: (SEL)

71518a00 dgram       0      0 183c3b80        0        0        0 /dev/.SRC-unix/SRCCwfCEb7051d400
         so_state: (PRIV)
         timeo:0 uid:0
         so_special: (LOCKBALE)
         so_special2: (PROC)
         sndbuf:
                 hiwat:16384 lowat:4096 mbcnt:0 mbmax:65536
         rcvbuf:
                 hiwat:8192 lowat:1 mbcnt:0 mbmax:32768
                 sb_flags: (SEL)

[...]
```

この例では、アダプターはジャンボ・フレーム用に構成されています。MSS の値が大きいのはこのためであり、**rfc1323** が設定されているのもこのためです。

netstat -M コマンド*:*

netstat -M コマンドは、ネットワーク・メモリーのクラスター・プールの統計情報を表示します。

次に、**netstat -M** コマンドの出力例を示します。

```
# netstat -M
Cluster pool Statistics:

  Cluster Size   Pool Size   Calls   Failed   Inuse   Max Outcount
        131072           0       0        0       0              0
         65536           0       0        0       0              0
         32768           0       0        0       0              0
         16384           0       0        0       0              0
          8192           0  191292        3       0              3
          4096           0  196021        3       0              3
          2048           0  140660        4       0              2
          1024           0       2        1       0              1
```

```
   512          0       2        1       0             1
131072          0       0        0       0             0
 65536          0       0        0       0             0
 32768          0       0        0       0             0
 16384          0       0        0       0             0
  8192          0  193948        2       0             2
  4096          0  191122        3       0             3
  2048          0  145477        4       0             2
  1024          0       0        0       0             0
   512          0       2        1       0             1
```

netstat -v コマンド*:*

netstat -v コマンドは、操作中の各共通データ・リンク・インターフェース (CDLI) ベースのデバイス・ドライバーの統計情報を示します。

tokstat、**entstat**、**fddistat**、または **atmstat** コマンドを使用すると、インターフェース固有のレポートを要求できます。

すべてのインターフェースには、独自の特定情報と多少の一般情報があります。次の例は、**netstat -v** コマンドのトークンリングとイーサネット部分です。その他のインターフェース部分は同様です。 アダプターが異なると、統計情報も多少異なります。 最も重要な出力フィールドは強調表示されます。

```
# netstat -v
-------------------------------------------------------------
ETHERNET STATISTICS (ent1) :
Device Type: 10/100 Mbps Ethernet PCI Adapter II (1410ff01)
Hardware Address: 00:09:6b:3e:00:55
Elapsed Time: 0 days 17 hours 38 minutes 35 seconds

Transmit Statistics:                          Receive Statistics:
--------------------                          -------------------
Packets: 519                                  Packets: 30161
Bytes: 81415                                  Bytes: 7947141
Interrupts: 2                                 Interrupts: 29873
Transmit Errors: 0                            Receive Errors: 0
Packets Dropped: 0                            Packets Dropped: 0
                                              Bad Packets: 0
Max Packets on S/W Transmit Queue: 3
S/W Transmit Queue Overflow: 0
Current S/W+H/W Transmit Queue Length: 1

Broadcast Packets: 3                          Broadcast Packets: 29544
Multicast Packets: 2                          Multicast Packets: 42
No Carrier Sense: 0                           CRC Errors: 0
DMA Underrun: 0                               DMA Overrun: 0
Lost CTS Errors: 0                            Alignment Errors: 0
Max Collision Errors: 0                       No Resource Errors: 0
Late Collision Errors: 0                      Receive Collision Errors: 0
Deferred: 0                                   Packet Too Short Errors: 0
SQE Test: 0                                   Packet Too Long Errors: 0
Timeout Errors: 0                             Packets Discarded by Adapter: 0
Single Collision Count: 0                     Receiver Start Count: 0
Multiple Collision Count: 0
Current HW Transmit Queue Length: 1

General Statistics:
-------------------
No mbuf Errors: 0
Adapter Reset Count: 0
Adapter Data Rate: 200
Driver Flags: Up Broadcast Running
        Simplex AlternateAddress 64BitSupport
        ChecksumOffload PrivateSegment DataRateSet
```

```
10/100 Mbps Ethernet PCI Adapter II (1410ff01) Specific Statistics:
--------------------------------------------------------------------
Link Status: Up
Media Speed Selected: Auto negotiation
Media Speed Running: 100 Mbps Full Duplex
Receive Pool Buffer Size: 1024
Free Receive Pool Buffers: 1024
No Receive Pool Buffer Errors: 0
Receive Buffer Too Small Errors: 0
Entries to transmit timeout routine: 0
Transmit IPsec packets: 0
Transmit IPsec packets dropped: 0
Receive IPsec packets: 0
Receive IPsec packets dropped: 0
Inbound IPsec SA offload count: 0
Transmit Large Send packets: 0
Transmit Large Send packets dropped: 0
Packets with Transmit collisions:
 1 collisions: 0           6 collisions: 0          11 collisions: 0
 2 collisions: 0           7 collisions: 0          12 collisions: 0
 3 collisions: 0           8 collisions: 0          13 collisions: 0
 4 collisions: 0           9 collisions: 0          14 collisions: 0
 5 collisions: 0          10 collisions: 0          15 collisions: 0
-------------------------------------------------------------
ETHERNET STATISTICS (ent0) :
Device Type: 10/100/1000 Base-TX PCI-X Adapter (14106902)
Hardware Address: 00:02:55:6a:a5:dc
Elapsed Time: 0 days 17 hours 0 minutes 26 seconds

Transmit Statistics:                          Receive Statistics:
--------------------                          -------------------
Packets: 15                                   Packets: 14
Bytes: 1037                                   Bytes: 958
Interrupts: 0                                 Interrupts: 13
Transmit Errors: 0                            Receive Errors: 0
Packets Dropped: 0                            Packets Dropped: 0
                                              Bad Packets: 0
Max Packets on S/W Transmit Queue: 4
S/W Transmit Queue Overflow: 0
Current S/W+H/W Transmit Queue Length: 0

Broadcast Packets: 1                          Broadcast Packets: 0
Multicast Packets: 1                          Multicast Packets: 0
No Carrier Sense: 0                           CRC Errors: 0
DMA Underrun: 0                               DMA Overrun: 0
Lost CTS Errors: 0                            Alignment Errors: 0
Max Collision Errors: 0                       No Resource Errors: 0
Late Collision Errors: 0                      Receive Collision Errors: 0
Deferred: 0                                   Packet Too Short Errors: 0
SQE Test: 0                                   Packet Too Long Errors: 0
Timeout Errors: 0                             Packets Discarded by Adapter: 0
Single Collision Count: 0                     Receiver Start Count: 0
Multiple Collision Count: 0
Current HW Transmit Queue Length: 0

General Statistics:
-------------------
No mbuf Errors: 0
Adapter Reset Count: 0
Adapter Data Rate: 2000
Driver Flags: Up Broadcast Running
        Simplex 64BitSupport ChecksumOffload
        PrivateSegment LargeSend DataRateSet

10/100/1000 Base-TX PCI-X Adapter (14106902) Specific Statistics:
--------------------------------------------------------------------
```

```
Link Status: Up
Media Speed Selected: Auto negotiation
Media Speed Running: 1000 Mbps Full Duplex
PCI Mode: PCI-X (100-133)
PCI Bus Width: 64-bit
Jumbo Frames: Disabled
TCP Segmentation Offload: Enabled
        TCP Segmentation Offload Packets Transmitted: 0
        TCP Segmentation Offload Packet Errors: 0
Transmit and Receive Flow Control Status: Enabled
        XON Flow Control Packets Transmitted: 0
        XON Flow Control Packets Received: 0
        XOFF Flow Control Packets Transmitted: 0
        XOFF Flow Control Packets Received: 0
Transmit and Receive Flow Control Threshold (High): 32768
Transmit and Receive Flow Control Threshold (Low): 24576
Transmit and Receive Storage Allocation (TX/RX): 16/48
```

強調表示されているフィールドについて、以下に説明します。

- **Transmit and Receive Errors**

  このデバイスで発生した出力/入力エラーの数。 このフィールドでは、ハードウェア/ソフトウェアのエラーが原因で成功しなかった送信をカウントします。

  成功しなかった送信は、システムのパフォーマンスを低下させることもあります。

- **Max Packets on S/W Transmit Queue**

  ソフトウェア送信キューにこれまで入れられた発信パケットの最大数。

  キュー・サイズが不十分であることが示されたら、キューに入れられた送信の最大数が現在のキュー・サイズ (*xmt_que_size*) と等しくなっています。 これは、ある時点でキューが満杯になったことを示しています。

  現在のキュー・サイズを調べるには、**lsattr -El** *adapter* コマンドを使用します (このアダプターは、例えば `ent0` などです)。キューがインターフェースのデバイス・ドライバーとアダプターに関連付けられている場合は、インターフェース名ではなく、アダプター名を使用してください。 キュー・サイズを変更するには、SMIT または **chdev** コマンドを使用します。

- **S/W Transmit Queue Overflow**

  ソフトウェア送信キューをオーバーフローさせた発信パケットの数。 ゼロ以外の値は、「`Max Packets on S/W Transmit Queue`」が *xmt_que_size* に達した場合と同じアクションが必要であることを示しています。 送信キューのサイズを大きくする必要があります。

- **Broadcast Packets**

  エラーなしで受け取ったブロードキャスト・パケットの数。

  ブロードキャスト・パケットの値が大きい場合は、受け取ったパケットの合計数と比較してください。受け取ったブロードキャスト・パケットは、受け取ったパケットの合計数の 20% 未満でなければなりません。 それより大きい場合は、ネットワーク・ロードが高いことを示しているので、マルチキャスティングを使用してください。 IP マルチキャスティングを使用すると、メッセージをホストのグループへ送信できるので、各グループ・メンバーへ個別にメッセージをアドレッシングし、送信する必要がありません。

- **DMA Overrun**

「DMA Overrun」統計情報は、アダプターが DMA を使用してシステム・メモリーにパケットを書き込んでいる最中で、転送が完了していないときに増分されます。 パケットを書き込むシステム・バッファーはありますが、DMA 操作が完了に失敗しました。 これは、アダプターがパケットのために DMA を使用できないほど MCA バスがビジー状態になっているときに起きます。バス上のアダプターの位置は、過度に負荷がかかるシステムでは重要なことです。一般的に、バス上でスロット番号が小さい位置にあるアダプターは、バス優先順位が高く、バスの多くを占有するので、高位スロット番号のアダプター用にバスが残されないことがあります。 これは特に、低位スロット番号のアダプターが ATM アダプターである場合に該当します。

- **Max Collision Errors**

  衝突が多すぎて成功しなかった送信の数。 検出された衝突の数が、アダプターの再試行回数を超えました。

- **Late Collision Errors**

  遅延衝突エラーが原因で成功しなかった送信の数。

- **Timeout Errors**

  タイムアウト・エラーが報告されたアダプターが原因で成功しなかった送信の数。

- **Single Collision Count**

  送信時に単一 (1 つだけの) 衝突が検出された発信パケットの数。

- **Multiple Collision Count**

  送信時に複数 (2 から 15 個の) 衝突が検出された発信パケットの数。

- **Receive Collision Errors**

  受信時に衝突エラーがあった着信パケットの数。

- **No mbuf Errors**

  デバイス・ドライバーが mbuf を使用できなかった回数。 これは通常、ドライバーがメモリー・バッファーを取得して到着パケットを処理する必要があるときに、受信操作中に起こります。 要求されたサイズの mbuf プールが空であるときには、パケットが廃棄されます。 **netstat -m** コマンドを使用してこれを確認し、パラメーター *thewall* の値を大きくしてください。

  「No mbuf Errors」の値は、インターフェース固有で、**netstat -m** の出力に示される「requests for mbufs denied」とは異なります。 **netstat -m** コマンドと **netstat -v** コマンド (イーサネットおよびトークンリング部分) の例に示されている値を比較してください。

ネットワーク・パフォーマンス上の問題を判別するには、**netstat -v** の出力に示されている「Error」カウントを調べてください。

その他のガイドライン:

- イーサネット・ネットワークの過負荷を調べるには、次のように計算します (**netstat -v** コマンドの出力から値を取り出します)。

  ```
  (Max Collision Errors + Timeouts Errors) / Transmit Packets
  ```

  結果が 5% を超える場合は、ネットワークを再編成して、負荷のバランスを取ってください。
- 次の場合も、ネットワークの負荷が高くなっています (**netstat -v** コマンドの出力から値を取り出します)。

次のように計算して、**netstat -v** 出力 (イーサネットに関する出力) に示された衝突の合計数が、送信されたパケットの合計数の 10% を超えている場合です。

```
Number of collisions / Number of Transmit Packets > 0.1
```

***netstat -p** プロトコル:*

netstat -p プロトコルは、プロトコル変数 (udp、tcp、sctp、ip、icmp) に指定された値に関する統計情報を表示します。この変数はプロトコルの既知の名前または別名です。

一部のプロトコル名と別名は、/etc/protocols ファイルにリストされています。 null 応答は、報告する数が存在しないことを示します。 その統計情報ルーチンが存在しない場合は、プロトコル変数に指定された値のプログラム・レポートが不明です。

次に、ip プロトコルの出力例を示します。

```
# netstat -p ip
ip:
        45775 total packets received
        0 bad header checksums
        0 with size smaller than minimum
        0 with data size < data length
        0 with header length < data size
        0 with data length < header length
        0 with bad options
        0 with incorrect version number
        0 fragments received
        0 fragments dropped (dup or out of space)
        0 fragments dropped after timeout
        0 packets reassembled ok
        45721 packets for this host
        51 packets for unknown/unsupported protocol
        0 packets forwarded
        4 packets not forwardable
        0 redirects sent
        33877 packets sent from this host
        0 packets sent with fabricated ip header
        0 output packets dropped due to no bufs, etc.
        0 output packets discarded due to no route
        0 output datagrams fragmented
        0 fragments created
        0 datagrams that can't be fragmented
        0 IP Multicast packets dropped due to no receiver
        0 successful path MTU discovery cycles
        1 path MTU rediscovery cycle attempted
        3 path MTU discovery no-response estimates
        3 path MTU discovery response timeouts
        1 path MTU discovery decrease detected
        8 path MTU discovery packets sent
        0 path MTU discovery memory allocation failures
        0 ipintrq overflows
        0 with illegal source
        0 packets processed by threads
        0 packets dropped by threads
        0 packets dropped due to the full socket receive buffer
        0 dead gateway detection packets sent
        0 dead gateway detection packet allocation failures
        0 dead gateway detection gateway allocation failures
```

強調表示されているフィールドについて、以下に説明します。

- **Total Packets Received**

  受け取った IP データグラムの合計数。

- **Bad Header Checksum or Fragments Dropped**

  「dup or out of space」が原因で出力に「bad header checksum」または「fragments dropped」と示されている場合は、ネットワークがパケットを破壊しているか、またはデバイス・ドライバー受信キューの大きさが十分ではありません。

- **Fragments Received**

  受け取ったフラグメントの合計数。

- **Dropped after Timeout**

  「fragments dropped after timeout」がゼロ以外の場合は、データグラムのすべてのフラグメントが到着する前に、ビジー状態のネットワークが原因で、ip フラグメントの「time to life counter」が経過しました。 これを回避するには、**no** コマンドを使用して *ipfragttl* ネットワーク・パラメーターの値を大きくします。 もう 1 つの理由として考えられるのは、mbuf の不足です。この場合は *thewall* の値を大きくしてください。

- **Packets Sent from this Host**

  このシステムで作成され、送信された IP データグラムの数。 このカウンターは、転送されたデータグラム (パススルー・トラフィック) を含みません。

- **Fragments Created**

  IP データグラムが送信されたときに、このシステムで作成されたフラグメントの数。

IP 統計情報を検討するときには、「fragments received」に対する「packets received」の比率を確認してください。 ネットワークの MTU が小さい場合のガイドラインとして、10% 以上のパケットがフラグメント化されている場合は、その原因を特定するために、さらに調査が必要です。 フラグメントの数が多い場合は、リモート・ホスト上で IP レイヤーの上位にあるプロトコルが、インターフェースの MTU よりデータのサイズが大きい IP へ、データを渡しています。 ネットワーク・パス内のゲートウェイ/ルーチンも、ネットワーク内のほかのノードより MTU サイズがかなり小さい可能性があります。 これと同じロジックを「packets sent」と「fragments created」にも適用できます。

フラグメント化が行われると CPU のオーバーヘッドが大きくなるので、その原因を特定することが重要です。 アプリケーションによっては、その性質上、フラグメント化が起きるものもあります。 例えば、少量のデータを送信するアプリケーションでは、フラグメント化が行われます。 ただし、アプリケーションが大量のデータを送信する場合でも、フラグメント化が起きる場合は、その原因を特定する必要があります。 多くの場合、使用される MTU のサイズは、システムで構成された MTU のサイズではありません。

次に、udp プロトコルの出力例を示します。

```
# netstat -p udp
udp:
        11623 datagrams received
        0 incomplete headers
        0 bad data length fields
        0 bad checksums
        620 dropped due to no socket
        10989 broadcast/multicast datagrams dropped due to no socket
        0 socket buffer overflows
        14 delivered
        12 datagrams output
```

注目すべき統計情報は次のとおりです。

- **Bad Checksums**

  ハードウェア・カードまたはケーブルのカードが原因で、チェックサムが無効になることがあります。

- **Dropped Due to No Socket**

  受け取った UDP データグラムのうち、宛先ソケット・ポートが開かれなかった数。 結果として、「ICMP Destination Unreachable - Port Unreachable」というメッセージが送信されます。 ただし、受け取った UDP データグラムがブロードキャスト・データグラムである場合は、ICMP エラーが生成されません。 この値が大きい場合は、アプリケーションがソケットをどのように処理しているかを調べてください。

- **Socket Buffer Overflows**

  ソケット・バッファーのオーバーフローは、送信および受信 UDP ソケットが不十分であること、**nfsd** デーモンが少なすぎること、または *nfs_socketsize*、*udp_recvspace* および *sb_max* の値が小さすぎることが原因であると考えられます。

netstat -p udp コマンドによってソケットのオーバーフローが示された場合は、サーバー上の **nfsd** デーモンの数を増やす必要があります。 まず、影響を受けるシステムを CPU または入出力の飽和状態について調べ、**no -a** コマンドを使用してその他の通信レイヤーの推奨設定値を調べます。 システムが飽和状態の場合は、その負荷を減らすか、またはリソースを増やす必要があります。

次に、tcp プロトコルの出力例を示します。

```
 # netstat -p tcp
tcp:
        576 packets sent
                512 data packets (62323 bytes)
                0 data packets (0 bytes) retransmitted
                55 ack-only packets (28 delayed)
                0 URG only packets
                0 window probe packets
                0 window update packets
                9 control packets
                0 large sends
                0 bytes sent using largesend
                0 bytes is the biggest largesend
        719 packets received
                504 acks (for 62334 bytes)
                19 duplicate acks
                0 acks for unsent data
                449 packets (4291 bytes) received in-sequence
                8 completely duplicate packets (8 bytes)
                0 old duplicate packets
                0 packets with some dup. data (0 bytes duped)
                5 out-of-order packets (0 bytes)
                0 packets (0 bytes) of data after window
                0 window probes
                2 window update packets
                0 packets received after close
                0 packets with bad hardware assisted checksum
                0 discarded for bad checksums
                0 discarded for bad header offset fields
                0 discarded because packet too short
                0 discarded by listeners
                0 discarded due to listener's queue full
                71 ack packet headers correctly predicted
                172 data packet headers correctly predicted
        6 connection requests
        8 connection accepts
        14 connections established (including accepts)
```

```
6 connections closed (including 0 drops)
0 connections with ECN capability
0 times responded to ECN
0 embryonic connections dropped
504 segments updated rtt (of 505 attempts)
0 segments with congestion window reduced bit set
0 segments with congestion experienced bit set
0 resends due to path MTU discovery
0 path MTU discovery terminations due to retransmits
0 retransmit timeouts
        0 connections dropped by rexmit timeout
0 fast retransmits
        0 when congestion window less than 4 segments
0 newreno retransmits
0 times avoided false fast retransmits
0 persist timeouts
        0 connections dropped due to persist timeout
16 keepalive timeouts
        16 keepalive probes sent
        0 connections dropped by keepalive
0 times SACK blocks array is extended
0 times SACK holes array is extended
0 packets dropped due to memory allocation failure
0 connections in timewait reused
0 delayed ACKs for SYN
0 delayed ACKs for FIN
0 send_and_disconnects
0 spliced connections
0 spliced connections closed
0 spliced connections reset
0 spliced connections timeout
0 spliced connections persist timeout
0 spliced connections keepalive timeout
```

注目すべき統計情報は次のとおりです。

- Packets Sent
- Data Packets
- Data Packets Retransmitted
- Packets Received
- Completely Duplicate Packets
- Retransmit Timeouts

TCP 統計情報の場合は、送信されたパケットの数と再送されたデータ・パケットの数を比較してください。 再送されたパケットの数が、送信されたパケットの合計数の 10% から 15% を超える場合は、TCP でタイムアウトが起きています。この場合は、ネットワーク・トラフィック量が多すぎるために、タイムアウト前に肯定応答 (ACK) が戻りません。 受信ノード上のボトルネックまたは一般的なネットワーク上の問題によっても、TCP の再送が行われることがあります。これにより、ネットワーク・トラフィック量が多くなり、さらにネットワーク・パフォーマンス上の問題が大きくなります。

また、受信されたパケットの数、および完全に重複しているパケットの数を比較してください。 送信ノード上の TCP が、受信ノードから ACK を受け取る前にタイムアウトになると、パケットが再送されます。 パケットの重複は、受信ノードが最終的にすべての再送パケットを受け取ったときに発生します。 重複パケットの数が 10% から 15% を超えると、問題はやはりネットワーク・トラフィック量が多くなりすぎること、または受信ノードにボトルネックが生じることです。 パケットが重複すると、ネットワーク・トラフィック量が多くなります。

再送タイムアウトの値は、TCP がパケットを送信したのに、ACK を時間内に受け取らなかったときに発生します。 このとき、パケットは再送されます。 この値は続いて再送が行われるたびに増分されます。このように再送が連続すると、CPU 使用率が高くなり、受信ノードがパケットを受け取らなかった場合は、パケットは最終的に破棄されます。

netstat -s:

netstat -s コマンドは、各プロトコルの統計情報を示します (一方、**netstat -p** コマンドは、指定されたプロトコルの統計情報を示します)。

netstat -s コマンドは、次のプロトコルのみの統計を表示します。

- TCP
- UDP
- SCTP
- IP
- IPv6
- IGMP
- ICMP
- ICMPv6

netstat -s -s:

文書化されていない **-s -s** オプションを指定すると、**netstat -s** 出力のうちゼロ以外の値の行だけを示すので、エラー・カウントを探しやすくなります。

netstat -s -Z:

この **netstat** コマンドは、**netstat -s** コマンドの統計カウンターをすべてクリアしてゼロにします。

netstat -r:

パフォーマンスに関係のあるもう 1 つのオプションは、発見されたパス最大伝送単位 (PMTU) の表示です。 この値を表示するには、**netstat -r** コマンドを使用してください。

2 つのホストが複数のネットワークのパスを経由して通信する場合は、送信されたパケットのサイズがパス内のどのネットワークの最小 MTU より大きいとき、パケットがフラグメント化されます。 パケットのフラグメント化によってネットワーク・パフォーマンスが低下することがあるので、ネットワーク・パス内の最小 MTU を超えないサイズのパケットを送信することによって、フラグメント化を回避することをお勧めします。 このサイズはパス *MTU* と呼ばれます。

次に、経路指定テーブルだけを表示するために **netstat -r -f inet** コマンドが使用される例を示します。

```
# netstat -r -f inet
Routing tables
Destination      Gateway           Flags   Refs     Use  If   PMTU Exp Groups

Route tree for Protocol Family 2 (Internet):
default          res101141         UGc        0        0  en1     -   -
ausdns01.srv.ibm  res101141        UGHW       1        4  en1  1500   -
10.1.14.0         server1           UHSb       0        0  en1     -   -  =>
10.1.14/24        server1           U          3      112  en1     -   -
brown17          loopback           UGHS       6      110  lo0     -   -
10.1.14.255       server1           UHSb       0        0  en1     -   -
magenta          res1031041         UGHW       1       42  en1     -   -
```

```
127/8            loopback         U         6    16633  lo0     -   -
192.1.6.0        en6host1         UHSb      0        0  en0     -   -  =>
192.1.6/24       en6host1         U         0       17  en0     -   -
en6host1         loopback         UGHS      0    16600  lo0     -   -
192.1.6.255      en6host1         UHSb      0        0  en0     -   -
192.6.0.0        fc0host1         UHSb      0        0  fc0     -   -  =>
192.6.0/24       fc0host1         U         0       20  fc0     -   -
fc0host1         loopback         UGHS      0        0  lo0     -   -
192.6.0.255      fc0host1         UHSb      0        0  fc0     -   -
```

netstat -D:

-D オプションを使用すると、パケットが通信サブシステムの各レイヤーに着信し、そこから発信されるのを確認できると共に、各レイヤーで破棄されるパケットも確認できます。

```
# netstat -D

Source                          Ipkts               Opkts     Idrops     Odrops
-------------------------------------------------------------------------------
ent_dev1                        32556                 727          0          0
ent_dev2                            0                   1          0          0
ent_dev3                            0                   1          0          0
fcnet_dev0                         24                  22          0          0
fcnet_dev1                          0                   0          0          0
ent_dev0                           14                  15          0          0
              ---------------------------------------------------------------
Devices Total                   32594                 766          0          0
-------------------------------------------------------------------------------
ent_dd1                         32556                 727          0          0
ent_dd2                             0                   2          0          1
ent_dd3                             0                   2          0          1
fcnet_dd0                          24                  22          0          0
fcnet_dd1                           0                   0          0          0
ent_dd0                            14                  15          0          0
              ---------------------------------------------------------------
Drivers Total                   32594                 768          0          2
-------------------------------------------------------------------------------
fcs_dmx0                            0                 N/A          0        N/A
fcs_dmx1                            0                 N/A          0        N/A
ent_dmx1                        31421                 N/A       1149        N/A
ent_dmx2                            0                 N/A          0        N/A
ent_dmx3                            0                 N/A          0        N/A
fcnet_dmx0                          0                 N/A          0        N/A
fcnet_dmx1                          0                 N/A          0        N/A
ent_dmx0                           14                 N/A          0        N/A
              ---------------------------------------------------------------
Demuxer Total                   31435                 N/A       1149        N/A
-------------------------------------------------------------------------------
IP                              46815               34058         64          8
IPv6                                0                   0          0          0
TCP                               862                 710          9          0
UDP                             12412                  13      12396          0
              ---------------------------------------------------------------
Protocols Total                 60089               34781      12469          8
-------------------------------------------------------------------------------
en_if1                          31421                 732          0          0
fc_if0                             24                  22          0          0
en_if0                             14                  20          0          6
lo_if0                          33341               33345          4          0
              ---------------------------------------------------------------
Net IF Total                    64800               34119          4          6
-------------------------------------------------------------------------------
(Note:  N/A -> Not Applicable)
```

「デバイス」レイヤーは、アダプターに着信するパケットの数、アダプターから発信されるパケットの数、および入出力時に破棄されたパケットの数を示します。 アダプターのエラーにはさまざまな原因があるので、その詳細については、**netstat -v** コマンドを調べる方法があります。

「ドライバー」レイヤーは、各アダプターのデバイス・ドライバーが処理したパケットのカウントを示します。 この場合は、カウントされたエラーを判別するために、**netstat -v** コマンドの出力が役立ちます。

「Demuxer」の値は、Demux レイヤーでカウントされたパケットを示し、この「Idrops」は通常、フィルター操作によってパケットがリジェクトされたこと (例えば、Netware または DecNet パケットが、調査中のシステムで処理されないためにリジェクトされた場合など) を表示します。

「プロトコル」レイヤーの詳細は、**netstat -s** コマンドの出力で確認できます。

注**:** 統計情報出力では、フィールド値に「N/A」と表示されることがあり、これはカウントが適用されないことを示します。 NFS/RPC 統計情報の場合は、RPC をパススルーした着信パケットの数が、NFS をパススルーしたパケットの数と同じなので、これらの数は「NFS/RPC Total」フィールドの合計に含まれず、「N/A」となります。NFS には、NFS と RPC に特定の発信パケットまたは発信パケット・ドロップのカウンターがありません。 したがって、個々のカウントのフィールド値は「N/A」となり、「NFS/RPC Total」フィールドには累積カウントが保管されます。

netpmon コマンド

netpmon コマンドでは、トレース機能を使用して、一定の時間内に行われたネットワーク・アクティビティーの詳細図を取得します。 **netpmon** コマンドは、トレース機能を使用するので、root ユーザーまたはシステム・グループのメンバーだけが実行できます。

netpmon コマンドは、**tprof** や **filemon** のような、他のトレース・ベースのパフォーマンス・コマンドと共に実行することはできません。 **netpmon** コマンドは、通常モードでは、1 つ以上のアプリケーション・プログラムまたはシステム・コマンドが実行およびモニターされているバックグラウンドで実行されます。

netpmon コマンドは、次のシステム・アクティビティーをフォーカスします。

- CPU 使用率
  - プロセスおよび割り込みハンドラーごとの使用率
  - ネットワーク関連の使用率
  - アイドル時間の原因
- ネットワーク・デバイス・ドライバーの入出力
  - イーサネット、トークンリング、および光ファイバー分散データ・インターフェース (FDDI) のすべてのネットワーク・デバイス・ドライバーによる入出力操作をモニターする。
  - 送信入出力の場合は、コマンドが使用率、キューの長さ、および宛先ホストをモニターする。 受信 ID の場合も、コマンドが Demux レイヤーで時間をモニターする。
- インターネット・ソケット・コール
  - インターネット・ソケット上で **send()**、**recv()**、**sendto()**、**recvfrom()**、**sendmsg()**、**read()**、および **write()** サブルーチンをモニターする。
  - ICMP (Internet Control Message Protocol)、TCP (Transmission Control Protocol)、および UDP (User Datagram Protocol) のプロセスごとの統計情報を報告する。
- NFS 入出力
  - クライアント上: RPC 要求、NFS 読み取り/書き込み要求

– サーバー上: クライアントごと、ファイルごと、読み取り/書き込み要求

次のものが計算されます。

- デバイス・ドライバー・レベルの送信操作と受信操作に関連する応答時間とサイズ
- インターネット・ソケットの読み取りおよび書き込みシステム・コールのすべてのタイプに関連する応答時間とサイズ
- NFS 読み取りおよび書き込みシステム・コールに関連する応答時間とサイズ
- NFS リモート・プロシージャー・コール要求に関連する応答時間

netpmon コマンドがインストールされているかどうか、および使用可能かどうかを判別するには、次のコマンドを実行します。

```
# lslpp -lI perfagent.tools
```

トレースは **netpmon** コマンドによって開始され、オプションで **trcoff** サブコマンドによって中断され、**trcon** サブコマンドによって再開されます。 トレースが終了するとすぐに、**netpmon** コマンドがそのレポートを標準出力に書き出します。

netpmon コマンドの使用法**:**

netpmon コマンドは、**-d** オプションが指定されている場合を除き、即時にトレースを開始します。

トレースを停止するには、**trcstop** コマンドを実行します。 **netpmon** コマンドは、停止されると、指定されたレポートをすべて生成してから、終了します。 クライアント/サーバー環境では、**netpmon** コマンドを使用すると、ネットワーキングがパフォーマンス全体に及ぼす影響を確認できます。 クライアントとサーバーのどちらでも実行できます。

netpmon コマンドは、リアルタイム・トレース・プロセスからではなく、指定されたファイルから入出力トレース・データを読み取ることができます。 この場合は、**netpmon** のレポートによって、トレース・ファイルが表すシステムと期間のネットワーク・アクティビティーが要約されます。このオフライン処理方式は、リモート・マシンからトレース・ファイルをポストプロセスする必要があるとき、またはトレース・データの収集を実行してから、また別のときにポストプロセスする必要があるときに、役立ちます。

trcrpt -r コマンドは、次のように、トレース・ログ・ファイル上で実行し、別のファイルへリダイレクトする必要があります。

```
# gennames > gennames.out
# trcrpt -r trace.out > trace.rpt
```

この時点で、調整されたトレース・ログ・ファイルが **netpmon** コマンドへ送られ、次のように以前に記録されたトレース・セッションで取り込まれた入出力アクティビティーが報告されます。

```
# netpmon -i trace.rpt -n gennames.out | pg
```

この例では、**netpmon** コマンドが `trace.rpt` 入力ファイルからファイルシステムのトレース・イベントを読み取ります。 トレース・データは既にファイルに取り込まれているため、**netpmon** コマンドは、アプリケーション・プログラムを実行できるようにするためにバックグラウンドに移ることはありません。ファイル全体が読み取られたら、標準出力にネットワーク・アクティビティー・レポートが表示されます (この例では、**pg** コマンドにパイピングされています)。

-C all フラグが指定されて **trace** コマンドが実行された場合は、同様に **-C all** フラグを指定して **trcrpt** コマンドを実行してください (434 ページの『トレース -C 出力によるレポートのフォーマット設定』を参照してください)。

次の **netpmon** コマンドを NFS サーバーで実行すると、**sleep** コマンドが実行され、 400 秒後にレポートが作成されます。 測定された時間内に、NFS マウントのファイルシステム /nfs_mnt にコピーされます。

```
# netpmon -o netpmon.out -O all; sleep 400; trcstop
```

-O オプションを指定すると、生成するレポートのタイプを指定できます。 有効なレポートのタイプ値は次のとおりです。

cpu CPU 使用率

dd ネットワーク・デバイス・ドライバー入出力

so インターネット・ソケット・コール入出力

nfs NFS 入出力

all すべてのレポートを生成。 次がデフォルト値です。

```
# cat netpmon.out

Fri Mar  5 15:41:52 2004
System: AIX crusade Node: 5 Machine: 000353534C00


========================================================================

Process CPU Usage Statistics:
-----------------------------
                                                 Network
Process (top 20)             PID  CPU Time   CPU %   CPU %
------------------------------------------------------------
netpmon                    45600    0.6995   1.023   0.000
nfsd                       50090    0.5743   0.840   0.840
UNKNOWN                    56912    0.1274   0.186   0.000
trcstop                    28716    0.0048   0.007   0.000
gil                         3870    0.0027   0.004   0.004
ksh                        42186    0.0024   0.003   0.000
IBM.ServiceRMd             14966    0.0021   0.003   0.000
IBM.ERrmd                   6610    0.0020   0.003   0.000
IBM.CSMAgentRMd            15222    0.0020   0.003   0.000
IBM.AuditRMd               12276    0.0020   0.003   0.000
syncd                       4766    0.0020   0.003   0.000
sleep                      28714    0.0017   0.002   0.000
swapper                        0    0.0012   0.002   0.000
rpc.lockd                  34942    0.0007   0.001   0.000
netpmon                    28712    0.0006   0.001   0.000
trace                      54622    0.0005   0.001   0.000
reaper                      2580    0.0003   0.000   0.000
netm                        3612    0.0002   0.000   0.000
aixmibd                     4868    0.0001   0.000   0.000
xmgc                        3354    0.0001   0.000   0.000
------------------------------------------------------------
Total (all processes)               1.4267   2.087   0.844
Idle time                          55.4400  81.108

========================================================================

First Level Interrupt Handler CPU Usage Statistics:
---------------------------------------------------
                                                 Network
FLIH                               CPU Time   CPU %   CPU %
------------------------------------------------------------
external device                      0.3821   0.559   0.559
PPC decrementer                      0.0482   0.070   0.000
data page fault                      0.0137   0.020   0.000
queued interrupt                     0.0002   0.000   0.000
```

```
------------------------------------------------------------
Total (all FLIHs)                   0.4441   0.650   0.559

================================================================================

Second Level Interrupt Handler CPU Usage Statistics:
---------------------------------------------------
                                                    Network
SLIH                              CPU Time   CPU %   CPU %
------------------------------------------------------------
phxentdd32                          2.4740   3.619   3.619
------------------------------------------------------------
Total (all SLIHs)                   2.4740   3.619   3.619

================================================================================

Network Device-Driver Statistics (by Device):
---------------------------------------------
                        ----------- Xmit -----------   -------- Recv ---------
Device                   Pkts/s  Bytes/s  Util  QLen   Pkts/s  Bytes/s   Demux
-------------------------------------------------------------------------------
ethernet 4              7237.33 10957295  0.0%27.303  3862.63   282624  0.2324

================================================================================

Network Device-Driver Transmit Statistics (by Destination Host):
----------------------------------------------------------------

Host                     Pkts/s  Bytes/s
-----------------------------------------
client_machine           7237.33 10957295

================================================================================

NFS Server Statistics (by Client):
----------------------------------
                         ------ Read -----   ----- Write -----     Other
Client                   Calls/s   Bytes/s   Calls/s   Bytes/s   Calls/s
-------------------------------------------------------------------------
client_machine      0.00         0      0.00         0    321.15
-------------------------------------------------------------------------
Total (all clients)         0.00         0      0.00         0    321.15

================================================================================

Detailed Second Level Interrupt Handler CPU Usage Statistics:
-------------------------------------------------------------

SLIH: phxentdd32
count:                  33256
  cpu time (msec):      avg 0.074   min 0.018   max 288.374 sdev 1.581

COMBINED (All SLIHs)
count:                  33256
  cpu time (msec):      avg 0.074   min 0.018   max 288.374 sdev 1.581

================================================================================

Detailed Network Device-Driver Statistics:
------------------------------------------

DEVICE: ethernet 4
recv packets:           33003
  recv sizes (bytes):   avg 73.2    min 60      max 618     sdev 43.8
  recv times (msec):    avg 0.000   min 0.000   max 0.005   sdev 0.000
  demux times (msec):   avg 0.060   min 0.004   max 288.360 sdev 1.587
xmit packets:           61837
```

```
  xmit sizes (bytes):   avg 1514.0  min 1349    max 1514    sdev 0.7
  xmit times (msec):    avg 3.773   min 2.026   max 293.112 sdev 8.947

========================================================================

Detailed Network Device-Driver Transmit Statistics (by Host):
-------------------------------------------------------------

HOST: client_machine (10.4.104.159)
xmit packets:           61837
  xmit sizes (bytes):   avg 1514.0  min 1349    max 1514    sdev 0.7
  xmit times (msec):    avg 3.773   min 2.026   max 293.112 sdev 8.947

========================================================================

Detailed NFS Server Statistics (by Client):
-------------------------------------------

CLIENT: client_machine
other calls:            2744
  other times (msec):   avg 0.192   min 0.075   max 0.311   sdev 0.025

COMBINED (All Clients)
other calls:            2744
  other times (msec):   avg 0.192   min 0.075   max 0.311   sdev 0.025
```

netpmon コマンドの出力は、2 種類のレポートで構成されています。1 つはグローバル で、もう 1 つは詳細 です。グローバル・レポートには、次のような統計情報がリストされます。

- 最もアクティブなプロセス
- 第 1 レベル割り込みハンドラー
- 第 2 レベル割り込みハンドラー
- ネットワーク・デバイス・ドライバー
- ネットワーク・デバイス・ドライバーの送信
- TCP ソケット・コール
- NFS サーバーまたはクライアントの統計情報

グローバル・レポートは、**netpmon** 出力の最初に示され、その内容は測定された間隔内のオカレンスです。 詳細レポートでは、グローバル・レポートの追加情報が提供されます。 デフォルトでは、測定された統計情報のうち最もアクティブな 20 の統計情報のみにレポートが限定されます。 レポート内の情報は、最もアクティブなものから順に、上から下へリストされます。

netpmon コマンドのグローバル・レポート**:**

netpmon コマンドによって生成されたレポートは、日付、マシン ID、およびモニター期間 (秒単位) を識別するヘッダーで始まります。

ヘッダーの次に、指定されたすべてのレポート・タイプのグローバル・レポートと詳細レポートのセットが続きます。

マイクロプロセッサー使用率の統計情報**:**

各行には、プロセスに関連するマイクロプロセッサー使用率が記述されます。

冗長 (**-v**) オプションが指定されない限り、最もアクティブな 20 個のプロセスがリストされます。 レポートの一番下には、すべてのプロセスのマイクロプロセッサー使用率が合計され、マイクロプロセッサーの

アイドル時間が報告されます。アイドル時間のパーセンテージは、測定された時間間隔で除算されたアイドル時間から計算されます。 マイクロプロセッサー時間の合計と測定された間隔との差は、割り込みハンドラーによるものです。

「Network CPU %」は、このプロセスがネットワーク関連のコードを実行するのに要した時間の合計に対するパーセンテージです。

-t フラグが使用された場合は、スレッド・マイクロプロセッサー使用率の統計情報も報告されます。 上記の各プロセス行の次には、そのプロセスが所有する各スレッドのマイクロプロセッサー使用率を記述する行が続きます。この行のフィールドは、名前フィールドを除いて、プロセスのフィールドと同じです。 スレッドには名前がありません。

レポートの例では、グローバル・マイクロプロセッサー使用率レポートに示されている「Idle time」のパーセンテージ (81.104%) は、「Idle time」(55.4400) を「measured interval」に 8 を掛けた数 (8.54 秒 x 8) で除算した値です。これは、このサーバーにマイクロプロセッサーが 8 つあるためです。 各マイクロプロセッサーのアクティビティーを確認するには、**sar**、**ps**、またはその他の SMP 固有コマンドを使用します。 これと同様の計算が、すべてのプロセスが占有する合計「CPU %」に適用されます。 「Idle time」はネットワーク入出力によるものです。「CPU Time」の合計 (55.4400 + 1.4267) と「measured interval」との差は、割り込みハンドラーおよび複数のマイクロプロセッサーによるものです。 レポートの例では、マイクロプロセッサー使用率の大部分がネットワーク関連で、(0.844 / 2.087) = 40.44% となっています。

注**:** 合計ネットワーク「CPU %」を合計「CPU %」で割った結果が、NFS サーバーの「Process CPU Usage Statistics」の 0.5 より大きい場合は、マイクロプロセッサー使用率の大部分がネットワーク関連です。

この方法もまた、特定のプログラムに出力せずにプロセスのマイクロプロセッサー使用率を確認できるよい方法です。

第 ***1*** レベル割り込みハンドラー・マイクロプロセッサー使用率の統計情報*:*

各行には、第 1 レベル割り込みハンドラー (FLIH) に関連するマイクロプロセッサー使用率が記述されます。

レポートの一番下には、すべての FLIH のマイクロプロセッサー使用率の合計が示されます。

CPU Time
この FLIH が使用するマイクロプロセッサーの合計時間

CPU %
合計時間に対するこの割り込みハンドラーのマイクロプロセッサー使用率のパーセンテージ

Network CPU %
ネットワーク関連のイベントの代わりにこの割り込みハンドラーが実行した合計時間のパーセンテージ

第 ***2*** レベル割り込みハンドラー・マイクロプロセッサー使用率の統計情報*:*

各行には、第 2 レベル割り込みハンドラー (SLIH) に関連するマイクロプロセッサー使用率が記述されます。 レポートの一番下には、すべての SLIH のマイクロプロセッサー使用率の合計が示されます。

デバイスごとのネットワーク・デバイス・ドライバーの統計情報*:*

netpmon コマンドは、デバイスごとのネットワーク・デバイス・ドライバーの統計情報をリストするレポートを作成する場合に使用します。

各行には、ネットワーク・デバイスに関連する統計情報が記述されます。

Device
デバイスに関連する特殊ファイルの名前

Xmit Pkts/s
このデバイスを介して送信されたパケット数/秒

Xmit Bytes/s
このデバイスを介して送信されたバイト数/秒

Xmit Util
このデバイスのビジー時間 (合計時間に対するパーセンテージ)

Xmit Qlen
このデバイスを介して送信されるのを待機している要求の数 (現在送信されているトランザクションを含む) で、時間の平均を取ったもの

Recv Pkts/s
このデバイスを介して受け取ったパケット数/秒

Recv Bytes/s
このデバイスを介して受け取ったバイト数/秒

Recv Demux
Demux レイヤーで費やされた時間 (合計時間に占める割合で表される)

この例では、「Xmit QLen」は 27.303 です。 その「Recv Bytes/s」は 10957295 (10.5 MB/秒) であり、これは 100 Mbps イーサネットでのワイヤー限界に近い数値です。 したがって、この場合のネットワークは、ほぼ飽和状態です。

宛先ホストごとのネットワーク・デバイス・ドライバーの伝送統計情報*:*

netpmon コマンドは、宛先ホストごとのネットワーク・デバイス・ドライバーの伝送統計情報をリストするレポートを作成する場合に使用します。

各行には、デバイス・ドライバー・レベルの、特定の宛先ホストに関連する送信トラフィックの量が記述されます。

Host 宛先ホスト名。 アスタリスク (*) は、ホスト名を判別できない送信を表すために使用されます。

Pkts/s このホストに送信されたパケット数/秒

Bytes/s
このホストに送信されたバイト数/秒

プロセスによる *IP* ごとの *TCP* ソケット呼び出しの統計情報*:*

これらの統計情報は、使用されたインターネット・プロトコルごとに示されます。

各行には、特定のプロセスに関連するこのプロトコル・タイプのソケット上での **read()** および **write()** サブルーチンのアクティビティーが記述されます。レポートの一番下には、このプロトコルのすべてのソケット・コールが合計されたものが示されます。

クライアントごとの *NFS* サーバーの統計情報*:*

各行には、特定のクライアントの代わりにこのサーバーが処理した NFS アクティビティーの量が記述されます。 レポートの一番下には、すべてのクライアントの呼び出しが合計されて示されます。

クライアント・マシン上では、NFS サーバー統計情報が、NFS クライアント統計情報 (「NFS Client Statistics for each Server (by File)」、「NFS Client RPC Statistics (by Server)」、「NFS Client Statistics (by Process)」) に置き換えられます。

netpmon の詳細レポート**:**

要求されたすべての **(-O)** レポート・タイプについて詳細レポートが生成されます。 これらのレポート・タイプについて、グローバル・レポートに加えて、詳細レポートが生成されます。 詳細レポートには、グローバル・レポート内の各エントリーごとのエントリー、およびそのエントリーに関連するトランザクションの各タイプについての統計情報が含まれます。

トランザクション統計情報には、そのタイプのトランザクションの数のカウントが示され、そのあとに応答時間とサイズ配分データ (存在する場合のみ) が続きます。 配分データは、平均、最小、および最大値からなり、標準偏差も示されます。 値の約 2/3 は、平均から標準偏差を引いた値から、平均に標準偏差を足した値までの範囲内です。サイズはバイト単位で報告されます。 応答時間は、ミリ秒単位で報告されます。

第 **2** レベル割り込みハンドラー・マイクロプロセッサー使用率の詳細な統計情報*:*

netpmon コマンドにより、第 2 レベル割り込みハンドラー・マイクロプロセッサー使用率の詳細な統計情報を表すレポートを作成できます。

出力フィールドについて、以下に説明します。

SLIH 第 2 レベル割り込みハンドラーの名前

count このタイプの割り込みの数

cpu time (msec)
このタイプの割り込みの処理に関するマイクロプロセッサー使用率の統計情報

デバイスごとのネットワーク・デバイス・ドライバーの詳細な統計情報*:*

netpmon コマンドにより、デバイス・ドライバーの各デバイスごとのネットワーク・デバイス・ドライバーの詳細な統計情報を表すレポートを作成できます。

出力フィールドについて、以下に説明します。

DEVICE
デバイスに関連付けられている特殊ファイルのパス名

recv packets
このデバイスを介して受け取ったパケットの数

recv sizes (bytes)
受け取ったパケットに関するサイズの統計情報

recv times (msec)
受け取ったパケットの処理に関する応答時間の統計情報

demux times (msec)
Demux レイヤーでの受け取ったパケットの処理に関する時間の統計情報

xmit packets
このデバイスを介して送信されたパケットの数

xmit sizes (bytes)
送信されたパケットに関するサイズの統計情報

xmit times (msec)
送信されたパケットの処理に関する応答時間の統計情報

このほかにも、詳細レポートがあります。例えば、「Detailed Network Device-Driver Transmit Statistics (by Host)」および「Detailed TCP Socket Call Statistics for Each Internet Protocol (by Process)」などです。 NFS クライアントの場合は、「Detailed NFS Client Statistics for Each Server (by File)」、「Detailed NFS Client RPC Statistics (by Server)」、および「Detailed NFS Client Statistics (by Process)」の各レポートがあります。 NFS サーバーの場合は、「Detailed NFS Server Statistics (by Client)」のレポートがあります。 これらの統計には、上記で説明したのと同様の出力フィールドがあります。

例では、「Detailed Network Device-Driver Statistics」から、次のことが分かります。

- 受信バイト数 = 33003 (パケット数) * 73.2 バイト/パケット = 2,415,819.6 バイト
- 送信バイト数 = 61837 (パケット数) * 1514 バイト/パケット = 93,621,218 バイト
- 交換された合計バイト数 = 2,415,819.6 + 93,621,218 = 96,037,037.6 バイト
- 交換された合計ビット数 = 96,037,037.6 * 8 ビット/バイト = 768,296,300.8 ビット
- ネットワークの速度 = 768,296,300.8 / 8.54 = 89,964,438 ビット/秒 (約 90 Mbps) - NFS コピーがトレースの全体量を占めていたと想定

グローバル・デバイス・ドライバー・レポートの場合と同様に、この場合も、ほぼネットワーク飽和状態であると言えます。 平均受信サイズの 73.2 バイトは、トレースされた NFS サーバーが、送信したデータの肯定応答を受信したということを示しています。 平均送信サイズは 1514 バイトであり、これはイーサネット・デバイス *interface* のデフォルトの MTU (maximum transmission unit) です。*interface* を en0 や tr0 などのインターフェース名に置き換えることにより、次のコマンドで、MTU またはアダプターの送信キューの長さの値を変更して、パフォーマンスを向上させることができます。

```
# ifconfig tr0 mtu 8500
```

または

```
# chdev -l 'tok0' -a xmt_que_size='150'
```

ネットワークが既に混雑している場合は、MTU またはキューの値を変更しても効果はありません。

注**:**

1. デバイス・ドライバー統計情報レポートの送信パケットおよび受信パケットのサイズが小さい場合は、現在の MTU サイズを大きくすると、ネットワークのパフォーマンスが向上する可能性があります。
2. NFS クライアント・レポートのネットワーク待機時間統計情報でネットワーク・コールによるシステム待ち時間が長い場合は、ネットワークが原因でパフォーマンスが低くなっています。

netpmon コマンドの制限**:**

netpmon コマンドでは、トレース機能を使用して統計情報を収集します。 したがって、次のようにシステムのワークロードに影響を与えます。

- 中ぐらいの規模の、ネットワーク中心のワークロードの場合は、**netpmon** コマンドによって、CPU 使用率全体が 3% から 5% 上昇する。
- 入出力は少ないのに飽和状態になっている CPU の環境では、**netpmon** コマンドによって、大規模コンパイルの速度が約 3.5% 低下する。

このような状態を緩和するには、オフライン処理を使用して、多くの CPU を備えたシステムで **-C all** フラグを指定して **trace** コマンドを実行します。

traceroute コマンド

traceroute コマンドは、ネットワークのテスト、測定、および管理に使用することを目的としたものです。

ping コマンドが IP ネットワークの到達可能性を確認しているときには、特定された問題の位置を正確に把握し、改善することはできません。 次の状態を考慮してください。

- システムと宛先の間にホップ (例えば、ゲートウェイや経路など) が多数存在し、パス上のいずれかの位置に問題があると考えられるとき。 宛先システムの問題である可能性がありますが、パケットが実際に失われた位置を確認する必要があります。
- **ping** コマンドがハングアップし、パケットが失われた理由が通知されない。

traceroute コマンドは、パケットの位置と経路が失われた理由を通知します。 ほかの組織または企業に属し、管理されているルーターとリンクをパケットが通過する必要がある場合は、**telnet** コマンドによって、関連するルーターを調べるのは困難です。 **traceroute** コマンドは、**ping** コマンドを補足します。

注: **traceroute** コマンドは、主に、手動での障害分離に使用されます。 ネットワークに負荷がかかるので、通常の操作または自動化されたスクリプトの実行中には **traceroute** コマンドを使用しないでください。

traceroute の成功例

traceroute コマンドは、UDP パケットを使用し、ICMP エラー報告機能を使用します。 経由する各ゲートウェイまたはルーターごとに 3 回ずつ、UDP パケットを送信します。 最も近いゲートウェイから開始し、1 つのホップによる検索を拡張します。 最後に、検索が宛先システムに到達します。 出力には、ゲートウェイ名、ゲートウェイの IP アドレス、およびゲートウェイの 3 つの往復時間が示されます。 次に例を示します。

```
# traceroute aix1
trying to get source for aix1
source should be 10.53.155.187
traceroute to aix1.austin.ibm.com (10.53.153.120) from 10.53.155.187 (10.53.155.187), 30 hops max
outgoing MTU = 1500
 1  10.111.154.1 (10.111.154.1)  5 ms  3 ms  2 ms
 2  aix1 (10.53.153.120)  5 ms  5 ms  5 ms
```

次にもう 1 つの例を示します。

```
# traceroute aix1
trying to get source for aix1
source should be 10.53.155.187
traceroute to aix1.austin.ibm.com (10.53.153.120) from 10.53.155.187 (10.53.155.187), 30 hops max
outgoing MTU = 1500
 1  10.111.154.1 (10.111.154.1)  10 ms  2 ms  3 ms
 2  aix1 (10.53.153.120)  8 ms  7 ms  5 ms
```

アドレス解決プロトコル (ARP) のエントリーの有効期限が切れたあと、同じコマンドが繰り返し実行されています。 各ゲートウェイまたは宛先に送られた最初のパケットは、往復する時間が長くかかっていることに注目してください。 これは、ARP によるオーバーヘッドが原因です。 経路に WAN (広域ネットワーク) が含まれている場合は、接続確立のために最初のパケットがメモリーの多くを占有し、タイムアウトとなる可能性があります。 各パケットのデフォルト・タイムアウトは 3 秒です。これは **-w** オプションで変更できます。

最初の 10 ミリ秒は、送信元システム (9.53.155.187) とゲートウェイ 9.111.154.1 間の ARP が原因となっています。次の 8 ミリ秒は、ゲートウェイと最終宛先 (wave) 間の ARP が原因となっています。 この場合は DNS を使用すると、**traceroute** コマンドがパケットを送信する前に必ず、DNS サーバーが検索されます。

traceroute の失敗例

宛先または複合ネットワーク経路までのパスが長い場合は、**traceroute** コマンドで多くの問題を発見できます。 多くの問題はインプリメンテーションに依存するので、問題の検索に時間がかかることがあるだけです。 関係しているすべてのルーターまたはシステムがユーザーの制御下にあるときには、多くの場合、問題を完全に調べることができます。

ゲートウェイ (ルーター) の問題

次の例では、パケットがシステム 9.53.155.187 から送信されています。 ブリッジまでの経路には 2 つのルーター・システムがあります。 2 番目のルーター・システムからは、**no** コマンドのオプション **ipforwarding** を 0 に設定することによって、経路指定機能を意図的に除去してあります。次に例を示します。

```
# traceroute lamar
trying to get source for lamar
source should be 9.53.155.187
traceroute to lamar.austin.ibm.com (9.3.200.141) from 9.53.155.187 (9.53.155.187), 30 hops max
outgoing MTU = 1500
 1  9.111.154.1 (9.111.154.1)  12 ms  3 ms  2 ms
 2  9.111.154.1 (9.111.154.1)  3 ms !H *  6 ms !H
```

ICMP エラー・メッセージ (Time Exceeded および Port Unreachable の 2 つのメッセージを除く) を受け取った場合は、次のように表示されます。

!H ホストが到達不可能

!N ネットワークが到達不可能

!P プロトコルが到達不可能

!S 送信元経路が失敗した

!F フラグメント化が必要

宛先システムの問題

宛先システムが 3 秒のタイムアウト間隔内に応答しない場合は、すべての照会がタイムアウトになり、結果がアスタリスク (*) で表示されます。

```
# traceroute chuys
trying to get source for chuys
source should be 9.53.155.187
traceroute to chuys.austin.ibm.com (9.53.155.188) from 9.53.155.187 (9.53.155.187), 30 hops max
outgoing MTU = 1500
 1  * * *
 2  * * *
 3  * * *
^C#
```

問題の原因が通信リンクであると考えられる場合は、**-w** フラグを指定してタイムアウト期間を長くします。 まれではありますが、照会されたすべてのポートが使用されている可能性があります。 ポートを変更し、再試行してください。

宛先の「ホップ」の数

もう 1 つの出力例を次に示します。

```
# traceroute mysystem.university.edu (129.2.130.22)
traceroute to mysystem.university.edu (129.2.130.22), 30 hops max
1 helios.ee.lbl.gov (129.3.112.1) 0 ms 0 ms 0 ms
2 lilac-dmc.university.edu (129.2.216.1) 39 ms 19 ms 39 ms
3 lilac-dmc.university.edu (129.2.215.1) 19 ms 39 ms 19 ms
4 ccngw-ner-cc.university.edu (129.2.135.23) 39 ms 40 ms 19 ms
5 ccn-nerif35.university.edu (129.2.167.35) 39 ms 39 ms 39 ms
6 csgw/university.edu (129.2.132.254) 39 ms 59 ms 39 ms
7 * * *
8 * * *
9 * * *
10 * * *
11 * * *
12 * * *
13 rip.university.EDU (129.2.130.22) 59 ms! 39 ms! 39 ms!
```

iptrace デーモンと ipreport および ipfilter コマンド

ネットワーク・アクティビティーを監視するためには、多くのツールが用意されています。 ツールには、オペレーティング・システムで実行されるものと、専用ハードウェアで実行されるものがあります。 あるツールは、ワークロードによって生成される LAN アクティビティーの詳細なパケット別記述を取得するのに使用できます。このツールは **iptrace** デーモンと **ipreport** コマンドの組み合わせです。

オペレーティング・システムのバージョン・バージョン 4 で **iptrace** デーモンを使用する場合は、`bos.net.tcp.server` ファイルセットが必要です。 このファイルセットには、**iptrace** デーモンが組み込まれており、さらにいくつかの役立つコマンド (例えば、**trpt** および **tcdump** コマンドなど) が組み込まれています。 **iptrace** デーモンは、root ユーザーだけが始動できます。

デフォルトでは、**iptrace** デーモンがすべてのパケットをトレースします。 オプション **-a** を使用すると、アドレス解決プロトコル (ARP) パケットを除外できます。その他のオプションでは、特定の送信元ホスト (**-s**)、宛先ホスト (**-d**)、またはプロトコル (**-p**) だけにトレースの有効範囲を限定できます。 **iptrace** デーモンはプロセッサー時間の多くを占めることがあるため、トレースするパケットを記述するときには、できる限り特定してください。

iptrace はデーモンなので、コマンド・ラインから直接始動するのではなく、**startsrc** コマンドを使用して **iptrace** デーモンを始動してください。 この方法により、制御とシャットダウンを正常に行うのが容易になります。一般的な例を次に示します。

```
# startsrc -s iptrace -a "-i en0 /home/user/iptrace/log1"
```

このコマンドは、**iptrace** デーモンを始動し、ギガビット・イーサネット・インターフェース `en0` 上のすべてのアクティビティーをトレースするように命令して、トレース・データを `/home/user/iptrace/log1` に入れます。デーモンを停止するには、次のように入力します。

```
# stopsrc -s iptrace
```

startsrc コマンドで **iptrace** デーモンを始動しなかった場合は、**ps** コマンドを使用して、そのプロセス ID を見つけ、**kill** コマンドで終了する必要があります。

ipreport コマンドは、ログ・ファイルのフォーマッターです。その出力は標準出力に書き出されます。 オプションを指定すると、RPC パケットの認識とフォーマット設定 (**-r**)、番号による各パケットの識別 (**-n**)、およびプロトコルを識別する 3 文字の文字列が含まれている各行のプレフィックス変換 (**-s**) が可能になります。 作成されたばかりの `log1` ファイル (root ユーザーが所有) をフォーマットするための一般的な **ipreport** コマンドは、次のとおりです。

```
# ipreport -ns log1 >log1_formatted
```

これにより、次の例と同様の一連のパケット・レポートが生成されます。 最初のパケットは、**ping** パケットの前半です。最も注目すべきフィールドは次のとおりです。

- ソース (SRC) ・ホスト・アドレスおよび宛先 (DST) ホスト・アドレス (どちらも小数点付き 10 進数および ASCII 形式)
- IP パケット長 (ip_len)
- 使用中のプロトコルのうち最高レベルのプロトコル (ip_p)

```
Packet Number 7
ETH: ====( 98 bytes transmitted on interface en0 )==== 10:28:16.516070112
ETH:    [ 00:02:55:6a:a5:dc -> 00:02:55:af:20:2b ]  type 800  (IP)
IP:     < SRC =       192.1.6.1 >  (en6host1)
IP:     < DST =       192.1.6.2 >  (en6host2)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=84, ip_id=1789, ip_off=0
IP:     ip_ttl=255, ip_sum=28a6, ip_p = 1 (ICMP)
ICMP:   icmp_type=8 (ECHO_REQUEST)  icmp_id=18058  icmp_seq=3

Packet Number 8
ETH: ====( 98 bytes received on interface en0 )==== 10:28:16.516251667
ETH:    [ 00:02:55:af:20:2b -> 00:02:55:6a:a5:dc ]  type 800  (IP)
IP:     < SRC =       192.1.6.2 >  (en6host2)
IP:     < DST =       192.1.6.1 >  (en6host1)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=84, ip_id=11325, ip_off=0
IP:     ip_ttl=255, ip_sum=366, ip_p = 1 (ICMP)
ICMP:   icmp_type=0 (ECHO_REPLY)  icmp_id=18058  icmp_seq=3
```

次の例は、**ftp** 操作のフレームです。 IP パケットがこの LAN の MTU のサイズ (1492 バイト) であることに注目してください。

```
Packet Number 20
ETH: ====( 1177 bytes transmitted on interface en0 )==== 10:35:45.432353167
ETH:    [ 00:02:55:6a:a5:dc -> 00:02:55:af:20:2b ]  type 800  (IP)
IP:     < SRC =       192.1.6.1 >  (en6host1)
IP:     < DST =       192.1.6.2 >  (en6host2)
IP:     ip_v=4, ip_hl=20, ip_tos=8, ip_len=1163, ip_id=1983, ip_off=0
IP:     ip_ttl=60, ip_sum=e6a0, ip_p = 6 (TCP)
TCP:    <source port=32873, destination port=20(ftp-data) >
TCP:    th_seq=623eabdc, th_ack=973dcd95
TCP:    th_off=5, flags<PUSH | ACK>
TCP:    th_win=17520, th_sum=0, th_urp=0
TCP: 00000000     69707472 61636520 322e3000 00008240     |iptrace 2.0....@|
TCP: 00000010     2e4c9d00 00000065 6e000065 74000053     |.L.....en..et..S|
TCP: 00000020     59535841 49584906 01000040 2e4c9d1e     |YSXAIXI....@.L..|
TCP: 00000030     c0523400 0255af20 2b000255 6aa5dc08     |.R4..U. +..Uj...|
TCP: 00000040     00450000 5406f700 00ff0128 acc00106     |.E..T......(....|
TCP: 00000050     01c00106 0208005a 78468a00 00402e4c     |.......ZxF...@.L|
TCP: 00000060     9d0007df 2708090d 0a0b0c0d 0e0f1011     |....'...........|
TCP: 00000070     12131415 16171819 1a1b1c1d 1e1f2021     |.............. !|
TCP: 00000080     22232425 26272829 2a2b2c2d 2e2f3031     |"#$%&'()*+,-./01|
TCP: 00000090     32333435 36370000 0082402e 4c9d0000     |234567....@.L...|
             --------- Lots of uninteresting data omitted -----------
TCP: 00000440     15161718 191a1b1c 1d1e1f20 21222324     |........... !"#$|
TCP: 00000450     25262728 292a2b2c 2d2e2f30 31323334     |%&'()*+,-./01234|
TCP: 00000460     353637                                  |567             |
```

ipfilter コマンドは、**ipreport** 出力ファイルから各種操作のヘッダーを取り出し、それらをテーブルに表示します。要求と応答に関する一部のカスタマイズ済み NFS 情報も提供されます。

ipfilter コマンドがインストールされているかどうか、および使用可能かどうかを判別するには、次のコマンドを実行します。

```
# lslpp -lI perfagent.tools
```

コマンドの例を次に示します。

```
# ipfilter log1_formatted
```

現在認識されている操作のヘッダーは、udp、nfs、tcp、ipx、icmp です。 **ipfilter** コマンドには、次のような 3 タイプのレポートがあります。

- 選択されたすべての操作のリストを表示する単一ファイル (ipfilter.all)。 テーブルには、 packet number、 Time、Source & Destination、Length、Sequence #、Ack #、Source Port、 Destination Port、 Network Interface、および Operation Type が表示されます。
- 選択された各ヘッダーの個々のファイル (ipfilter.udp、ipfilter.nfs、ipfilter.tcp、ipfilter.ipx、ipfilter.icmp)。格納される情報は、ipfilter.all と同じです。
- NFS 要求および応答に関して報告するファイル nfs.rpt。テーブルには、Transaction ID #、 Type of Request、Status of Request、Call Packet Number、Time of Call、Size of Call、Reply Packet Number、 Time of Reply、Size of Reply、および Elapsed millisecond between call and reply が表示されます。

アダプター統計情報

このセクションのコマンドからは、**netstat -v** コマンドと同等の出力が提供されます。これらのコマンドを使用すると、アダプター統計情報のリセット (**-r**) および **netstat -v** コマンドの出力で得られる出力より詳細な出力の取得 (**-d**) が可能です。

entstat コマンド

entstat コマンドは、指定されたイーサネット・デバイス・ドライバーが収集した統計情報を表示します。ユーザーは、デバイスの一般的な統計情報に加えて、デバイス固有の統計情報が表示されるように、オプションで指定できます。 **-d** オプションを使用すると、このアダプターの拡張統計情報がリストされるので、すべての統計情報を表示するときに使用してください。 フラグを指定しないと、デバイスの一般的な統計情報のみが表示されます。

entstat コマンドは、 **-v** フラグが指定されて **netstat** コマンドが実行されたときにも呼び出されます。**netstat** コマンドは、**entstat** コマンド・フラグを発行しません。

```
# entstat ent0
------------------------------------------------------------
ETHERNET STATISTICS (ent0) :
Device Type: 10/100/1000 Base-TX PCI-X Adapter (14106902)
Hardware Address: 00:02:55:6a:a5:dc
Elapsed Time: 1 days 18 hours 47 minutes 34 seconds

Transmit Statistics:                          Receive Statistics:
--------------------                          -------------------
Packets: 1108055                              Packets: 750811
Bytes: 4909388501                             Bytes: 57705832
Interrupts: 0                                 Interrupts: 681137
Transmit Errors: 0                            Receive Errors: 0
Packets Dropped: 0                            Packets Dropped: 0
                                              Bad Packets: 0
Max Packets on S/W Transmit Queue: 101
S/W Transmit Queue Overflow: 0
Current S/W+H/W Transmit Queue Length: 0

Broadcast Packets: 3                          Broadcast Packets: 3
Multicast Packets: 3                          Multicast Packets: 5
No Carrier Sense: 0                           CRC Errors: 0
DMA Underrun: 0                               DMA Overrun: 0
Lost CTS Errors: 0                            Alignment Errors: 0
Max Collision Errors: 0                       No Resource Errors: 0
```

```
Late Collision Errors: 0                    Receive Collision Errors: 0
Deferred: 0                                 Packet Too Short Errors: 0
SQE Test: 0                                 Packet Too Long Errors: 0
Timeout Errors: 0                           Packets Discarded by Adapter: 0
Single Collision Count: 0                   Receiver Start Count: 0
Multiple Collision Count: 0
Current HW Transmit Queue Length: 0

General Statistics:
-------------------
No mbuf Errors: 0
Adapter Reset Count: 0
Adapter Data Rate: 2000
Driver Flags: Up Broadcast Running
        Simplex 64BitSupport ChecksumOffload
        PrivateSegment LargeSend DataRateSet
```

上記のレポートでは、以下に注目します。

Transmit Errors
このデバイスで発生した出力エラーの数。 これは、ハードウェア/ソフトウェアのエラーが原因で成功しなかった送信のカウンターです。

Receive Errors
このデバイスで発生した入力エラーの数。 これは、ハードウェア/ソフトウェアのエラーが原因で成功しなかった受信のカウンターです。

Packets Dropped
デバイス・ドライバーは送信を受け入れたが、デバイスには (何らかの理由で) 送られなかったパケットの数。

Max Packets on S/W Transmit Queue
ソフトウェア送信キューにこれまで入れられた発信パケットの最大数。

S/W Transmit Queue Overflow
送信キューをオーバーフローさせた発信パケットの数。

No Resource Errors
リソースの欠落が原因でハードウェアが破棄した着信パケットの数。このエラーは通常、アダプター上の受信バッファーを使い果たしたことが原因で発生します。 一部のアダプターには、受信バッファーのサイズを変更できる構成可能パラメーターがあります。 チューニング情報については、デバイス構成属性 (または SMIT のヘルプ) を調べてください。

Single Collision Count/Multiple Collision Count
イーサネット・ネットワーク上の衝突の数。 これらの衝突についてはここで説明し、**netstat -i** コマンド出力の衝突欄について記述する箇所では説明しません。

この例では、「Receive Errors」がないため、イーサネット・アダプターが正常に動作しています。 受信エラーは、飽和状態になったネットワークがパケットを部分的に送信することによって引き起こされることがあります。 残りの部分的なパケットは最終的には正常に再送されますが、受信エラーとして記録されます。

「S/W Transmit Queue Overflow」エラーを受け取った場合は、「Max Packets on S/W Transmit Queue」の値が、このアダプターの送信キュー制限 (**xmt_que_size**) に一致します。

注: アダプターがソフトウェア送信キューをサポートしていない場合は、これらの値が「*hardware queue*」を表すことがあります。 送信キューのオーバーフローが起きた場合は、ドライバーのハードウェアまたはソフトウェア・キュー制限の値を大きくしてください。

受信リソースが十分にない場合は、「Packets Dropped:」に示されます。また、アダプターのタイプによっては、「Out of Rcv Buffers」や「No Resource Errors:」、あるいは同様のカウンターで示されることもあります。

経過時間には、統計情報が最後にリセットされてから経過したリアルタイムが表示されます。 統計情報をリセットするには、**entstat -r** *adapter_name* コマンドを使用してください。

トークンリング、FDDI、および ATM インターフェースの場合は、**tokstat**、**fddistat**、および **atmstat** コマンドを使用すると同様の出力を得られます。

tokstat コマンド

tokstat コマンドは、指定されたトークンリング・デバイス・ドライバーが収集した統計情報を表示します。ユーザーは、デバイス・ドライバーの統計情報に加えて、デバイス固有の統計情報が表示されるように、オプションで指定できます。 フラグを指定しないと、デバイス・ドライバーの統計情報のみが表示されます。

このコマンドは、**-v** フラグが指定されて **netstat** コマンドが実行されたときにも呼び出されます。 **netstat** コマンドは、**tokstat** コマンド・フラグを発行しません。

tokstat tok0 コマンドによって生成される出力と問題判別は、 353 ページの『entstat コマンド』の説明と同様です。

fddistat コマンド

fddistat コマンドは、指定された FDDI デバイス・ドライバーが収集した統計情報を表示します。 ユーザーは、デバイス・ドライバーの統計情報に加えて、デバイス固有の統計情報が表示されるように、オプションで指定できます。 フラグを指定しないと、デバイス・ドライバーの統計情報のみが表示されます。

このコマンドは、**-v** フラグが指定されて **netstat** コマンドが実行されたときにも呼び出されます。 **netstat** コマンドは、**fddistat** コマンド・フラグを発行しません。

fddistat fddi0 コマンドによって生成される出力と問題判別は、 353 ページの『entstat コマンド』の説明と同様です。

atmstat コマンド

atmstat コマンドは、指定された ATM デバイス・ドライバーが収集した統計情報を表示します。 ユーザーは、デバイス・ドライバーの統計情報に加えて、デバイス固有の統計情報が表示されるように、オプションで指定できます。 フラグを指定しないと、デバイス・ドライバーの統計情報のみが表示されます。

atmstat atm0 コマンドによって生成される出力と問題判別は、 353 ページの『entstat コマンド』の説明と同様です。

no コマンド

現在のネットワーク値の表示とオプションの変更は、**no** コマンドとこのコマンドのフラグを使用します。

-a すべてのオプションと現行値を出力します。

-d オプションをデフォルトに戻します。

-o *option=NewValue*

no コマンドのすべての属性をリストするには、 494 ページの『ネットワーク・オプションのチューナブル・パラメーター』を参照してください。

注**: no** コマンドは範囲検査を実行しません。 使用法を誤ると、**no** コマンドによってシステムが実行不能になることがあります。

ネットワーク属性のなかには、随時変更できるランタイム属性もあります。 ほかの属性はロード・タイム属性で、**netinet** カーネル・エクステンションをロードする前に設定する必要があります。

注**: no** コマンドを使用してパラメーターを変更した場合、変更が有効なのは次にシステムをブートするまでです。ブートした時点で、すべてのパラメーターがデフォルトにリセットされます。

注**:** 今後のリブートで特定の **no** コマンド・オプションを使用可能にしたり、使用不可にしたりするために、上記の情報が `/etc/tunables/nextboot` ファイルに書き込まれていなければなりません。これを行うには、コマンド・ラインで **no -r -o <***no_optionname***>=<***value***>** を入力します (例えば、**no -r -o arptab_bsiz=10**)。後続のリブートでは、**arptab_bsiz=10** が有効なまま存続し、nextboot ファイルに適用されます。

システムがバークレー・スタイルのネットワーク構成を採用している場合は、`/etc/rc.bsdnet` ファイルの一番上近くに属性を設定してください。 SP システムを使用する場合は、`tuning.cust` ファイルを編集してください。

NFS パフォーマンス

AIX は、サーバーとクライアントの両方で、ネットワーク・ファイルシステム (NFS) のモニターとチューニングを行うためのツールとメソッドを提供しています。

関連タスク:

499 ページの『NFS クライアントにおける大容量ファイルの書き込みパフォーマンスの向上』
NFS マウントされたファイルシステムに大きい順次ファイルを書き込むと、NFS サーバーへのファイル転送速度が大幅に低下する可能性があります。 このシナリオでは、この状況が起きているかどうかを特定し、この問題を修正する手順を使用します。

ネットワーク・ファイルシステム

NFS を使用すると、あるシステム上のプログラムが、リモート・ディレクトリーをマウントすることによって、別のシステム上のファイルへ透過的にアクセスできます。

通常、サーバーをブートすると、**exportfs** コマンドによってディレクトリーが使用可能になり、リモート・アクセスを処理するデーモン (**nfsd** デーモン) が始動されます。 同様に、クライアント・システムのブート時に、リモート・アクセスを処理するために、リモート・ディレクトリーのマウントと、適切な数の NFS ブロック入出力デーモン (**biod** デーモン) の開始が行われます。

nfsd および **biod** デーモンはいずれもマルチスレッドです。つまり、1 つのプロセスに複数のカーネル・スレッドがあります。 さらに、これらのデーモンは自己チューニング型であり、NFS アクティビティーの量に基づき、必要に応じてスレッドの作成や削除を自ら実行します。

次の図は、NFS クライアントとサーバー間のダイアログの構造を示しています。 クライアント・システム内のスレッドが NFS マウント・ディレクトリー内のファイルの読み取りあるいは書き込みを行おうとすると、要求は通常の入出力メカニズムから、クライアントの **biod** スレッドの 1 つへリダイレクトされます。 **biod** スレッドは、適切なサーバーへ要求を送信し、そのサーバーで、要求は、サーバーの NFS ス

レッド (**nfsd** スレッド) の 1 つに割り当てられます。その要求が処理されている間には、関与した **biod** スレッドも **nfsd** スレッドも、ほかの作業を行いません。

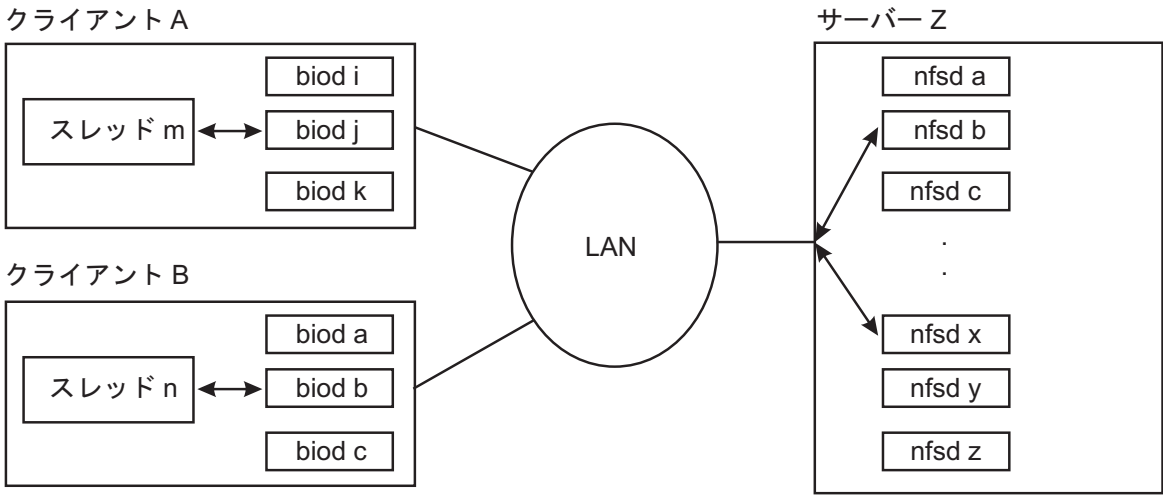


図 *22. NFS* クライアント/サーバーの対話： この図は、通常のスター型トポロジーで配置されたネットワーク上にある、2 つのクライアントと 1 つのサーバーを示しています。 クライアント A はアプリケーション・スレッド m を実行し、このスレッド内でデータは biod スレッドの 1 つに送られます。 同様にクライアント B はアプリケーション・スレッド n を実行し、データを biod スレッドの 1 つに送ります。 それぞれのスレッドは、データをネットワーク経由でサーバー Z に送信します。サーバー Z では、データはサーバーの NFS (nfsd) スレッドの 1 つに割り当てられます。

NFS は、リモート・プロシージャー・コール (RPC) を使用して、通信を行います。 RPC は、データを送信する前に汎用フォーマットに変換し、さまざまなアーキテクチャーをもったマシン間で情報を交換できるようにする外部データ表現 (XDR) プロトコルを使用して作成されます。 RPC ライブラリーは、ローカル (クライアント) プロセスが、自分自身のアドレス・スペースでプロシージャー・コールを実行したかのように、リモート (サーバー) プロセスにプロシージャー・コールを実行するように指示できるようにする、プロシージャーのライブラリーです。 クライアントとサーバーは 2 つの別々のプロセスなので、もはや同じ物理システム上に存在している必要はありません。

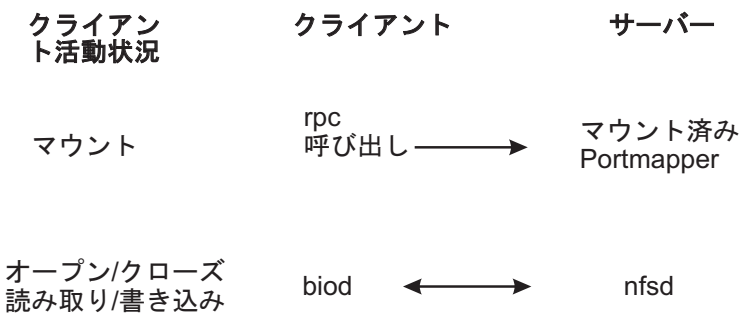


図 *23.* マウントおよび *NFS* プロセス： この図は、クライアントのアクティビティー、クライアント、およびサーバーの 3 つの列見出しがある 3 列の表です。 クライアントの最初のアクティビティーはマウントです。 RPC 呼び出しが、クライアントからサーバーの portmapper mountd に送られます。 クライアントの 2 番目のアクティビティーは、オープン/クローズ、読み取り/書き込みです。 クライアントの biod スレッドとサーバーの nfsd スレッドの間には、双方向の対話があります。

portmap デーモンである **portmapper** は、クライアントに、特定のプログラムに関連付けられているポート番号を検索するための標準の方法を提供する、ネットワーク・サーバー・デーモンです。 サーバー上のサービスは、要求されると、**portmap** デーモンに使用可能サーバーとして登録されます。 これで、**portmap** デーモンは、プログラムとポート間の対のテーブルを維持します。

クライアントは、サーバーに対する要求を開始すると、まず **portmap** デーモンに連絡して、サービスが存在している場所を調べます。 **portmap** デーモンは、予約済ポートで listen するので、クライアントは該当のサービスを探す必要がありません。 **portmap** デーモンは、クライアントが要求しているサービスのポートをクライアントに応答します。 クライアントは、ポート番号を受け取ったならば、それ以降の要求をすべて直接アプリケーションに出すことができるようになります。

mountd デーモンは、サーバーのエクスポート・ファイルシステムまたはディレクトリーをマウントするというクライアントの要求に応答するサーバー・デーモンです。 **mountd** デーモンは、 `/etc/xtab` ファイルを読み取ることによって、使用可能なファイルシステムを判別します。マウント・プロセスは、次のように行われます。

1. クライアントのマウントは、サーバーの **portmap** デーモンを呼び出して、 **mountd** デーモンに割り当てられたポート番号を見つけます。
2. **portmap** デーモンは、ポート番号をクライアントへ渡します。
3. クライアントの **mount** コマンドは次に、サーバーの **mountd** デーモンに直接連絡して、必要なディレクトリーの名前を渡します。
4. サーバーの **mountd** デーモンが、`/etc/xtab` (**exportfs -a** コマンドで作成されたもので、`/etc/exports` を読み取る) を調べて、要求されたディレクトリーについての可用性と許可を検証します。
5. すべてが検証されたならば、サーバーの **mountd** デーモンは、エクスポートされたディレクトリーのファイル・ハンドル (ファイルシステム・ディレクトリーを指すポインター) を取得し、それをクライアントのカーネルに渡します。

クライアントは、システム再始動後、最初のマウント要求時にだけ、**portmap** デーモンと連絡を取ります。 クライアントは、**mountd** デーモンのポート番号を認識したならば、それ以降のマウント要求のときには、そのポート番号と直接連絡を取ります。

biod デーモンは、ブロック入出力デーモンであり、ディレクトリー読み取りと同様に、先読みおよび遅延書き込み要求を実行するために必要です。 **biod** デーモン・スレッドは、NFS クライアント・アプリケーションの代わりに、バッファー・キャッシュを満杯にしたり、空にしたりすることによって、NFS パフォーマンスを向上させます。 クライアント・システム上のユーザーがサーバー上のファイルを読み取ったり、ファイルに書き込んだりするときには、**biod** スレッドがサーバーへ要求を送信します。 次の NFS オペレーションは、オペレーティング・システムの NFS クライアント・カーネル・エクステンションからサーバーへ直接送られるので、 **biod** デーモンを使用する必要はありません。

- **getattr()**
- **setattr()**
- **lookup()**
- **readlink()**
- **create()**
- **remove()**
- **rename()**
- **link()**

- **symlink()**
- **mkdir()**
- **rmdir()**
- **readdir()**
- **readdirplus()**
- **fsstat()**

nfsd デーモンは、NFS サーバーから NFS サービスを提供するアクティブ・エージェントです。 クライアントから 1 つの NFS プロトコル要求を受け取るときには、その要求が満たされ、要求処理の結果がクライアントに戻されるまで、 **nfsd** デーモン・スレッドが専心して注意を払う必要があります。

NFS ネットワーク・トランスポート

NFS のデフォルト・トランスポート・プロトコルは TCP ですが、UDP を使用することもできます。

トランスポート・プロトコルは、マウントごとに選択できます。 UDP は、クリーンであるかまたは効率的なネットワーク、および敏速に応答するサーバーでは効果的な働きをします。広域ネットワークや、ビジー・ネットワーク、またはサーバーが低速のネットワークの場合は、 TCP の本来のフロー制御によってネットワーク上の再送待ち時間が最小限になるので、 TCP の方がよりよいパフォーマンスを提供する可能性があります。

さまざまなバージョンの NFS

AIX は、NFS バージョン 2 とバージョン 3 の両方を同一のマシン上でサポートしています。このオペレーティング・システムは NFS バージョン 4 もサポートします。

AIX クライアントでマウント・オプションにバージョンが指定されていない場合は、引き続き NFS バージョン 3 がデフォルトとなります。ネットワーク・トランスポートの場合と同じく、マウントごとに NFS プロトコルのバージョンを選択できます。

NFS バージョン 4:

NFS バージョン 4 は NFS の最新のプロトコル仕様であり、RFC 3530 で定義されています。

このバージョンは以前のバージョンの NFS、特にバージョン 3 と類似していますが、新しいプロトコルにはセキュリティー、スケーラビリティー、およびバックエンド・データ管理の分野で新しい機能強化が数多く備わっています。 このような特性を備えた NFS バージョン 4 は、大規模分散ファイル共用環境に適しています。

NFS バージョン 4 プロトコルの重要な点には、以下のことがあります。

- 360 ページの『NFS 操作のインプリメンテーション変更』
- 360 ページの『TCP 要件』
- 360 ページの『ロック・プロトコルの統合』
- 360 ページの『マウント・サポートの統合』
- 360 ページの『セキュリティー機構の改良』
- 361 ページの『国際化対応サポート』
- 361 ページの『拡張可能属性モデル』
- 361 ページの『アクセス制御リスト・サポート』

新しいプロトコルで追加された機能や複雑さのために、処理オーバーヘッドが増えているので注意してください。 それで、多くのアプリケーションで、NFS バージョン 4 でのパフォーマンスが NFS バージョン 3 よりも遅い場合があります。 パフォーマンスの影響は、どの新しい機能を使用するかによってかなり異なります。 例えば、NFS バージョン 4 とバージョン 3 で同じセキュリティー機構を使用してみると、NFS バージョン 4 のシステムのほうが実行が若干遅い程度です。しかし、従来の UNIX 認証 (AUTH_SYS) を使用するバージョン 3 のパフォーマンスと Kerberos 5 にプライバシーを追加した (つまり完全ユーザー・データ暗号化) バージョン 4 のパフォーマンスを比較すると、パフォーマンスがかなり低下していることに気付くことでしょう。

また、NFS バージョン 3 用のすべてのチューニング推奨事項は、たいてい NFS バージョン 4 にも適用されます。

NFS 操作のインプリメンテーション変更*:*

NFS バージョン 2 および 3 とは異なり、バージョン 4 は 2 つの RPC プロシージャー NULL と COMPOUND のみで構成されています。

COMPOUND プロシージャーは、以前の NFS バージョンでは通常異なる RPC プロシージャーとして定義されていた 1 つ以上の NFS 操作で構成されています。 この変更により、ネットワークを介して論理ファイルシステム操作を実行するために必要とされる RPC が少なくなりました。

TCP 要件*:*

NFS バージョン 4 プロトコルでは、WAN 環境でのよりよいパフォーマンスを実現するために、輻輳 (ふくそう) 制御を含むトランスポート・プロトコルの使用が必須です。

AIX は、NFS バージョン 4 での UDP の使用をサポートしていません。

ロック・プロトコルの統合*:*

NFS バージョン 4 には、通知バイト範囲ファイル・ロックのサポートが組み込まれています。

ネットワーク・ロック・マネージャー (NLM) プロトコルおよび関連する `rpc.lockd` デーモンと `rpc.statd` デーモンは使用されていません。 UNIX 以外のオペレーティング・システムとのインターオペラビリティーを向上させるために、NFS バージョン 4 はオープンな共用予約もサポートしており、サーバー・プラットフォームに必須ロックを追加する機能も含まれています。

マウント・サポートの統合*:*

NFS バージョン 4 は、プロトコル操作を介したファイルシステムのマウントをサポートします。

クライアントが別個のマウント・プロトコルを使用したり、`rpc.mountd` デーモンを使って通信することはありません。

セキュリティー機構の改良*:*

NFS バージョン 4 には、RPCSEC-GSS セキュリティー・プロトコルのサポートが組み込まれています。

RPCSEC-GSS セキュリティー・プロトコルを使用することにより、新しい RPC 認証定義を 1 つ 1 つ追加せずに、複数のセキュリティー機構をインプリメンテーションできます。 AIX 上の NFS でサポートされているのは、Kerberos 5 セキュリティー機構だけです。

国際化対応サポート*:*

NFS バージョン 4 では、ストリング・ベースのデータはロー・バイトのまま送信されず、UTF-8 でエンコードされます。

拡張可能属性モデル*:*

NFS バージョン 4 の属性モデルを使用すると、UNIX 以外のインプリメンテーションとのインターオペラビリティーは向上し、ユーザーによる属性定義の追加も簡単に行えます。

アクセス制御リスト・サポート*:*

NFS バージョン 4 には ACL 属性の定義も組み込まれています。

ACL モデルは、ユーザーまたはグループ単位でアクセスを付与または否認するための許可やエントリー・タイプを設定できるという点で、Windows NT モデルと似ています。

NFS バージョン **3:**

NFS バージョン 3 は固有プロトコル機能が強化されてパフォーマンスを改善できるため、NFS バージョン 2 よりも強くお勧めします。

書き込みスループット*:*

クライアント・システムで実行中のアプリケーションは、定期的にファイルへデータを書き込み、ファイルの内容を変更することがあります。

アプリケーションが、一定時間にサーバー上の安定したストレージに書き込むデータ量が、分散ファイルシステムの書き込みスループット の測定基準です。 したがって、書き込みスループットは、パフォーマンスの重要な側面です。 NFS を含む分散ファイルシステムはすべて、データが安全に宛先ファイルに書き込まれることを保証する必要があるのと同時に、書き込みスループットのサーバー待ち時間による影響を最小限にすることも保証する必要があります。

NFS バージョン 3 のプロトコルは、NFS バージョン 2 の同期書き込み要求を除去することによって、クローズからオープンまでのセマンティクスの利点を保つ、書き込みスループットの増大に対する優れた代替策です。 NFS バージョン 3 クライアントは、データをディスクではなくサーバーのキャッシュ・ファイル・データ (メイン・メモリー) に書くことによって、サーバーに対する書き込み操作の待ち時間を大幅に削減しています。 その後、NFS クライアントはサーバーに対してコミット操作要求を出して、サーバーがすべてのデータを固定ストレージに書き込むようにします。 このフィーチャーは安全非同期書き込み と呼ばれ、サーバー上のディスク入出力要求の数を大幅に減らすので、書き込みスループットが著しく向上します。

書き込み時には、正常に保管されたかどうかを示すデータの状況情報が維持されるので、書き込みは「安全」であると見なされます。 したがって、サーバーがコミット・オペレーション前にクラッシュした場合、クライアントは、状況情報を調べることによって、サーバーが正常な状態に戻ったときに書き込み要求を再実行依頼すべきかどうかが分かります。

ファイル属性要求の減少*:*

読み取りデータは、要求を予測して、時間を延長してキャッシュ内に留まることがあるので、クライアントは、別のアプリケーションによってそのファイルに変更が加えられた場合に、そのキャッシュ・データがまだ有効であるかどうかを調べる必要があります。したがって、NFS クライアントは、定期的にファイルの

属性を取得します。ファイルの属性には、ファイルが最後に変更された時刻が含まれています。 この変更時刻によって、クライアントは、キャッシュ・データがまだ有効であるかどうかを判断できます。

属性要求を最小限にとどめると、クライアントの効率が上がり、サーバーの負荷が最小限に抑えられるので、スケーラビリティーとパフォーマンスが高くなります。 したがって、NFS バージョン 3 は、すべての操作で属性を戻すように設計されていました。 これにより、キャッシュ内の属性が最新のものになる可能性が高くなるので、個別の属性要求の数が減少するようになりました。

高帯域幅ネットワーク・テクノロジーの効率的な使用法*:*

RPC サイズ制限を緩和することにより、NFS は FDDI、100baseT (100 Mbps) および 1000baseT (ギガビット) イーサネット、SP スイッチなどの高帯域幅ネットワーク・テクノロジーを効果的に使用することができ、順次読み取り/書き込みでの NFS のパフォーマンスの改善に十分に貢献します。

NFS バージョン 2 には、最大 8 KB という RPC サイズ制限があるので、一度にネットワーク上で転送できる NFS データの量が制限されます。 NFS バージョン 3 では、この制限が緩和されています。 AIX の NFS では、デフォルトの読み取り/書き込みサイズが 32 KB で、最大は 64 KB なので、NFS は従来よりも大きい RPC パケット・データのチャンクを構成し、転送することができます。

ディレクトリー・ルックアップ要求の減少*:*

例えば、**ls -l** コマンドによって生成されるようなディレクトリーの完全なリストを作成するには、ディレクトリー・リストに入れるすべてのエントリーの属性情報をサーバーから取得する必要があります。

NFS バージョン 2 のクライアントは、ルックアップ要求内のすべてのディレクトリー・エントリーのファイル名とディレクトリー名のリスト、およびそれらの属性情報を取得するために、サーバーを別々に照会します。 NFS バージョン 3 では、名前リストと属性情報が **READDIRPLUS** オペレーションによって一度に戻されるので、クライアントとサーバーの両方が多数のタスクを処理する負荷を軽減できます。

名前ルックアップ・キャッシュ用の NFS クライアント・ディレクトリー (つまり **dnlc**) での長いファイル名 (31 文字を超える名前) のキャッシュに対するサポートが追加されました。このフィーチャーのインプリメンテーションにより、非常に長いファイル名を使用するときの NFS クライアントの作業負荷が軽減されます。以前は、長いファイル名を使用するときは、**dnlc** ミスのためにサーバーへの過度な **LOOKUP** オペレーションが余儀なくされていました。 このタイプの作業負荷の例は、前述の **ls -l** コマンドです。

NFS パフォーマンスのモニターおよびチューニング

NFS 統計情報をモニターするため、および NFS 属性をチューニングするために使用する幾つかのコマンドがあります。

良い NFS パフォーマンスを達成するには、チューニングを実行することに加えて、NFS 自身だけでなく、オペレーティング・システムやその基礎ハードウェアを含めたボトルネックを除去する必要があります。 高負荷の読み取り/書き込みアクティビティーを特色とするワークロードは特に重要で、システム全体のチューニングと構成が必要になります。 このセクションでは、 NFS の使用に適していないワークロードについても取り上げます。

一般的なルールでは、いかなるチューニング変数の調整を開始する前にも、これらの値を変更することで達成すべきものが何か、これらの変更が影響する可能性やマイナスの副次作用は何かについて理解していることを確認している必要があります。

NFS 統計情報とチューニング・パラメーター

NFS は、エラー情報とパフォーマンス・インディケーターと共に、実行された NFS オペレーションのタイプの統計情報を収集します。

次のコマンドを使用すると、潜在的なボトルネックを識別し、システムで発生した NFS オペレーションのタイプを監視したり、NFS 特有のパラメーターを調整できます。

nfsstat コマンド**:**

nfsstat コマンドは、NFS に関する統計情報と、クライアントおよびサーバーのカーネルに対する RPC インターフェースの統計情報を表示します。

このコマンドは、これらの統計情報のカウンターを初期化し直すために使用することもできます (**nfsstat -z**)。 パフォーマンス上の問題がある場合は、まず RPC 統計情報を調べてください (**-r** オプション)。 NFS 統計情報には、アプリケーションによる NFS の使用法が示されます。

RPC 統計情報**:**

nfsstat コマンドは、RPC コールに関する統計情報を表示します。

表示される統計情報のタイプは、以下のとおりです。

- 受信またはリジェクトされた RPC コールの合計数
- サーバーが送信またはリジェクトした RPC コールの合計数
- RPC パケットを受け取ろうとしたときに、入手できなかった回数
- 短すぎたか、またはヘッダーの形式が無効だったパケットの数
- コールをもう一度送信する必要があった回数
- 応答がコールに一致しなかった回数
- コールがタイムアウトになった回数
- コールがビジー状態のクライアント・ハンドルを待つ必要があった回数
- 認証情報を更新する必要があった回数

nfsstat コマンド出力の NFS 部分は、 NFS のバージョン 2 とバージョン 3 の統計情報に分けられます。 RPC 部分は、「Connection oriented」(TCP) と「Connectionless」(UDP) の統計情報に分けられます。

それぞれのトピックに特有の出力については、 372 ページの『サーバー上の NFS パフォーマンス・チューニング』および 376 ページの『クライアント上の NFS チューニング』を参照してください。

nfso コマンド**:**

nfso コマンドを使用して、NFS 属性を構成できます。

このコマンドでは、現在実行中のカーネルや NFS カーネル・エクステンションに関連する NFS 関連オプションの設定や表示を行います。 コマンドの詳細や出力については、*Commands Reference, Volume 4*の『nfso コマンド』を参照してください。

注**: nfso** コマンドは範囲チェックを実行しません。 使用の仕方が正しくないと、**nfso** コマンドはシステムを稼働できなくすることがあります。

nfso パラメーターとその値は、次のように **nfso -a** コマンドを使用することによって表示できます。

```
# nfso -a
                    portcheck = 0
                  udpchecksum = 1
               nfs_socketsize = 60000
            nfs_tcp_socketsize = 60000
             nfs_setattr_error = 0
          nfs_gather_threshold = 4096
           nfs_repeat_messages = 0
nfs_udp_duplicate_cache_size = 5000
nfs_tcp_duplicate_cache_size = 5000
    nfs_server_base_priority = 0
         nfs_dynamic_retrans = 1
            nfs_iopace_pages = 0
         nfs_max_connections = 0
             nfs_max_threads = 3891
      nfs_use_reserved_ports = 0
    nfs_device_specific_bufs = 1
           nfs_server_cread = 1
                nfs_rfc1323 = 1
          nfs_max_write_size = 65536
           nfs_max_read_size = 65536
       nfs_allow_all_signals = 0
                  nfs_v2_pdts = 1
                  nfs_v3_pdts = 1
               nfs_v2_vm_bufs = 1000
               nfs_v3_vm_bufs = 1000
   nfs_securenfs_authtimeout = 0
   nfs_v3_server_readdirplus = 1
           lockd_debug_level = 0
           statd_debug_level = 0
           statd_max_threads = 50
             utf8_validation = 1
                 nfs_v4_pdts = 1
              nfs_v4_vm_bufs = 1000
```

ほとんどの NFS 属性は、いつでも変更できる実行時属性です。 ロード時属性 (例えば、*nfs_socketsize* など) の場合は、まず NFS を停止し、あとから再始動する必要があります。 **nfso -L** コマンドは、現在の値、デフォルト値、値の変更が実際に有効になる時点に関する制限など、それぞれの属性の詳細な情報を提供します。

```
# nfso –L

NAME                      CUR    DEF    BOOT   MIN    MAX    UNIT           TYPE     DEPENDENCIES
--------------------------------------------------------------------------------
portcheck                 0      0      0      0      1      On/Off           D
--------------------------------------------------------------------------------
udpchecksum               1      1      1      0      1      On/Off           D
--------------------------------------------------------------------------------
nfs_socketsize            600000 600000 600000 40000  1M     Bytes            D
--------------------------------------------------------------------------------
nfs_tcp_socketsize        600000 600000 600000 40000  1M     Bytes            D
--------------------------------------------------------------------------------
nfs_setattr_error         0      0      0      0      1      On/Off           D
--------------------------------------------------------------------------------
nfs_gather_threshold      4K     4K     4K     512    8193   Bytes            D
--------------------------------------------------------------------------------
nfs_repeat_messages       0      0      0      0      1      On/Off           D
--------------------------------------------------------------------------------
nfs_udp_duplicate_cache_size
                          5000   5000   5000   5000   100000 Req              I
--------------------------------------------------------------------------------
nfs_tcp_duplicate_cache_size
                          5000   5000   5000   5000   100000 Req              I
--------------------------------------------------------------------------------
nfs_server_base_priority  0      0      0      31     125    Pri              D
--------------------------------------------------------------------------------
```

```
nfs_dynamic_retrans       1      1      1      0      1      On/Off            D
--------------------------------------------------------------------------------
nfs_iopace_pages          0      0      0      0      65535  Pages             D
--------------------------------------------------------------------------------
nfs_max_connections       0      0      0      0      10000  Number            D
--------------------------------------------------------------------------------
nfs_max_threads           3891   3891   3891   5      3891   Threads           D
--------------------------------------------------------------------------------
nfs_use_reserved_ports    0      0      0      0      1      On/Off            D
--------------------------------------------------------------------------------
nfs_device_specific_bufs  1      1      1      0      1      On/Off            D
--------------------------------------------------------------------------------
nfs_server_clread         1      1      1      0      1      On/Off            D
--------------------------------------------------------------------------------
nfs_rfc1323               1      1      0      0      1      On/Off            D
--------------------------------------------------------------------------------
nfs_max_write_size        64K    32K    32K    512    64K    Bytes             D
--------------------------------------------------------------------------------
nfs_max_read_size         64K    32K    32K    512    64K    Bytes             D
--------------------------------------------------------------------------------
nfs_allow_all_signals     0      0      0      0      1      On/Off            D
--------------------------------------------------------------------------------
nfs_v2_pdts               1      1      1      1      8      PDTs              M
--------------------------------------------------------------------------------
nfs_v3_pdts               1      1      1      1      8      PDTs              M
--------------------------------------------------------------------------------
nfs_v2_vm_bufs            1000   1000   1000   512    5000   Bufs              I
--------------------------------------------------------------------------------
nfs_v3_vm_bufs            1000   1000   1000   512    5000   Bufs              I
--------------------------------------------------------------------------------
nfs_securenfs_authtimeout 0      0      0      0      60     Seconds           D
--------------------------------------------------------------------------------
nfs_v3_server_readdirplus 1      1      1      0      1      On/Off            D
--------------------------------------------------------------------------------
lockd_debug_level         0      0      0      0      10     Level             D
--------------------------------------------------------------------------------
statd_debug_level         0      0      0      0      10     Level             D
--------------------------------------------------------------------------------
statd_max_threads         50     50     50     1      1000   Threads           D
--------------------------------------------------------------------------------
utf8_validation           1      1      1      0      1      On/Off            D
--------------------------------------------------------------------------------
nfs_v4_pdts               1      1      1      1      8      PDTs              M
--------------------------------------------------------------------------------
nfs_v4_vm_bufs            1000   1000   1000   512    5000   Bufs              I
--------------------------------------------------------------------------------


n/a means parameter not supported by the current platform or kernel

Parameter types:
    S = Static: cannot be changed
    D = Dynamic: can be freely changed
    B = Bosboot: can only be changed using bosboot and reboot
    R = Reboot: can only be changed during reboot
    C = Connect: changes are only effective for future socket connections
    M = Mount: changes are only effective for future mountings
    I = Incremental: can only be incremented

Value conventions:
    K = Kilo: 2^10       G = Giga: 2^30       P = Peta: 2^50
    M = Mega: 2^20       T = Tera: 2^40       E = Exa: 2^60
```

特定のパラメーターを表示または変更するには、**nfso -o** コマンドを使用します。次に例を示します。

```
# nfso -o portcheck
portcheck= 0
# nfso -o portcheck=1
```

パラメーターをデフォルト値にリセットするには、**-d** オプションを使用します。次に例を示します。

```
# nfso -d portcheck
# nfso -o portcheck
portcheck= 0
```

NFS パフォーマンスのための TCP/IP チューニング・ガイドライン

ネットワーク入出力を実行する場合は、NFS は UDP または TCP を使用します。

279 ページの『TCP および UDP のパフォーマンスのチューニング』および 316 ページの『mbuf プールのパフォーマンスのチューニング』に記述されているチューニング手法を必ず活用してください。 特に、次に挙げることを行ってください。

- **errpt** コマンドを実行し、ネットワーク・デバイスまたはネットワーク・メディアの問題に関するレポートを探して、システム・エラー・ログ・エントリーをチェックしてください。
- LAN アダプターの送信および受信キューを最大に設定してください。 詳しくは、307 ページの『アダプター・リソースのチューニング』を参照してください。
- **netstat -i** コマンドを使用して `Oerrs` をチェックしてください。これらのエラーが多数存在すれば、関連するネットワーク・デバイスの送信キューのサイズの大きさが十分でない可能性があります。
- TCP および UDP ソケット・バッファーのサイズが適切に構成されていることを確認してください。 **nfso** コマンドのチューニング・オプション *nfs_tcp_socketsize* は、NFS によって使用される TCP ソケット・バッファー・サイズ *tcp_sendspace* および *tcp_recvspace* を制御します。 同様に、チューニング・オプション *nfs_udp_socketsize* は、NFS で使用される UDP ソケット・バッファー・サイズ *udp_sendspace* および *udp_recvspace* を制御します。 ソケット・バッファー・サイズのチューニング・オプションを設定するために、 TCP および UDP パフォーマンス・チューニングに記述されているガイドラインに従ってください。 通常の TCP および UDP のチューニングの場合と同じく、**no** コマンドの **sb_max** チューニング・オプションの値は、*nfs_tcp_socketsize* および *nfs_udp_socketsize* の値よりも大きい必要があります。一般的には、AIX で使用されるデフォルト値は十分の値になっているはずですが、そのチェックは無駄ではありません。 UDP ソケット・バッファー・オーバーランがないか確認するために、 **netstat –s –p udp** コマンドを実行し、`socket buffer overflows` フィールドに多数のパケット破棄がレポートされているかどうかを確認します。
- システムに十分なネットワーク・メモリーが構成されていることを確認します。 **netstat –m** コマンドを実行して、拒否されたかまたは遅れた mbuf に対する要求が存在するかどうかを確認してください。存在する場合は、ネットワークに使用できる mbuf の数を増やしてください。 mbuf の問題を除去するためのシステムのチューニングの詳細については、 316 ページの『mbuf プールのパフォーマンスのチューニング』を参照してください。
- 一般的な経路指定の問題がないか調べてください。 **traceroute** コマンドを使用して、予期しない経路指定のホップまたは遅延を探してください。
- 可能であれば、LAN 上の MTU サイズを大きくします。 例えば、16 MB ギガビット・イーサネットでは、MTU サイズをデフォルトの 1500 バイトから 9000 バイト (ジャンボ・フレーム) まで大きくすると、 8 KB NFS 読み取りまたは書き込み要求が、フラグメント化せずに完全に送信されます。 また、mbuf スペースの使用もよりはるかに効果的になり、オーバーランの確率も減ります。
- MTU サイズ・ミスマッチがないか調べてください。 **netstat -i** コマンドを実行し、クライアントおよびサーバー上の MTU を調べてください。 2 つの MTU が異なる場合は、同じ値にするよう試行して、問題が除去されるかどうか確認してください。 また、マシン間のルーターやブリッジなどの低速あるいは広域ネットワーク装置が、パケットをさらにフラグメント化して、これらのネットワーク・セグ

メントを通過させることがあるので注意してください。 考えられる解決策として、送信元と宛先間の最小 MTU を判別し、NFS マウント上の **rsize** および **wsize** 設定を、最小公分母の MTU より小さい値に変更してください。

- NFS バージョン 3 を TCP で実行し、デフォルトの 32 KB 以上の RPC サイズを使用する場合、**nfso** コマンドの *nfs_rfc1323* オプションを設定する必要があります。 そうすれば、TCP ウィンドウ・サイズを 64 KB より大きい値にできるので、TCP 肯定応答待ちの最小化に役立ちます。 このオプションは、TCP 接続の両サイドで設定する必要があります。例えば、NFS サーバーとクライアントの両方で設定するということです。
- 非常に小さいパケット間の遅延を調べてください。 これによって問題が生じるのは、ごくまれなケースです。 サーバーとクライアント間にルーターまたはその他のハードウェアが存在する場合は、そのハードウェアの文書を調べて、パケット間の遅延を構成できるかどうかを確認してください。 構成できる場合は、遅延時間を長くしてみてください。
- 大きなメディア速度ミスマッチがないか調べてください。 パケットが、速度がまったく異なる 2 つのメディアを通過するときには、ルーターがパケットを高速ネットワークから取り出し、低速ネットワークにのせて送り出そうとするときに、パケットが破棄される可能性があります。例えば、ルーターがギガビット・イーサネット上のサーバーからパケットを取得し、それらを 100Mbps イーサネット上のクライアントへ送信しようとしたときに起こる可能性があります。ルーターは、ギガビット・イーサネットに追い付ける速さで、パケットを 100Mbps イーサネットで送信することはできません。 ルーターを置き換える以外に考えられる解決策は、クライアント要求の速度をスローダウンさせることや、読み取り/書き込みサイズを小さくすることです。
- サーバーに許される TCP 接続の最大数は、新しいオプション *nfs_max_connections* によって制御できます。 デフォルトの 0 は、制限がないことを示します。 クライアントは、約 5 分間アイドル状態だった TCP 接続を閉じ、使用によって妥当と認められたときに接続が再確立されます。 サーバーは、約 6 分間アイドル状態だった接続を閉じます。
- オペレーティング・システムは、NFS だけの UDP チェックサムをオフにするオプションを提供します。 *udpchecksum* という名前の **nfso** コマンド・オプションを使用できます。デフォルトは 1 で、これは、チェックサムが使用可能であることを意味します。 これをオフにすると、少しだけパフォーマンスを向上させることができますが、データが破壊される可能性が大きくなります。

破棄されたパケット

破棄されたパケットは最初に NFS クライアント上で検出されるのが一般的ですが、ここでの問題は、破棄された場所を突き止めることです。 パケットは、クライアント、サーバー、およびネットワーク上のどこかで破棄される可能性があります。

クライアントによって破棄されるパケット**:**

パケットはクライアントによって破棄されることはほとんどありません。

それぞれの **biod** スレッドは、一度に 1 つの NFS オペレーションしか処理できないため、他の RPC コールの実行前にオペレーションからの RPC コールの応答を待つ必要があります。 この自己ペース機構によって、システム・リソースのオーバーランの可能性はほとんどなくなります。 最も負荷が多いと考えられるオペレーションはおそらく読み取りであり、データが高速でマシンに流入する可能性があります。 データ・ボリュームは大きいことがありますが、同時発生する RPC コールの実際の数は小さく、各 **biod** スレッドには、応答のために各自のスペースが割り当てられます。 したがって、クライアントがパケットを破棄することは、極めてまれです。

パケットは、ネットワークまたはサーバーによって破棄される方が、より一般的です。

サーバーによって破棄されるパケット:

サーバーが過負荷の状況下でパケットを破棄する幾つかの状態があります。

1. ネットワーク・アダプター・ドライバー

   NFS サーバーが大量の要求に応答するときには、サーバーがインターフェース・ドライバーの出力キューをオーバーランすることがあります。 これは、**netstat -i** コマンドによって報告される統計情報を調べると監視できます。 「Oerrs」の欄にあるカウントを調べてください。 各「Oerrs」値は破棄されたパケットを表します。 これは、問題のデバイス・ドライバーの送信キューのサイズを大きくすることによって、容易にチューニングできます。構成可能キューの背後にある考え方は、キューの処理に要する待ち時間を考慮して、送信キューをあまり長くしないというものです。 しかし、NFS はコールのために同一のポートと XID を維持するので、最初のコールに対する応答によって、2 番目のコールが満たされます。 さらに、パケットが破棄されると、キュー処理の待ち時間は、NFS による UDP 再送待ち時間よりはるかに短くなります。

2. ソケット・バッファー

   UDP ソケット・バッファーは、サーバーがパケットを破棄するもう 1 つの場所です。 ここで破棄されたパケットは、UDP レイヤーによってカウントされ、**netstat -p udp** コマンドを使用することによって、統計情報を入手できます。 `socket buffer overflows` 統計情報を調べてください。

   NFS パケットは通常、サーバーの NFS 書き込みトラフィック量が多いときにだけ、ソケット・バッファーで破棄されます。 NFS サーバーは、NFS ポート 2049 に接続された UDP ソケットを使用し、着信データはすべて、その UDP ポート上でバッファーに入れられます。 このバッファーのデフォルトのサイズは 60,000 バイトです。 このバイト数を、デフォルトの NFS バージョン 3 書き込みパケットのサイズ (32786) で割ると、そのバッファーをオーバーフローさせるには、19 個の同時発生書き込みパケットが必要であることが分かります。

   サーバーがチューニングされ、破棄されたパケットがソケット・バッファーにも `Oerrs` ドライバーにも到達しないが、クライアントでは依然としてタイムアウトと再送が確認される場合があります。 この場合も、2 つの事例があります。 サーバーの負荷が大きい場合は、おそらくサーバーが単に過負荷であり、サーバー上の **nfsd** デーモンの作業のバックログによって、応答時間が、クライアントに設定されたデフォルト・タイムアウトを超えます。 もう 1 つの可能性としては、サーバーがそれ以外はアイドル状態であると認識されている場合に最も起こりやすい問題として、パケットがネットワーク上で破棄されている場合があります。

ネットワーク上で破棄されるパケット:

ソケット・バッファーのオーバーフローもサーバー上の「Oerrs」も検出されず、クライアントが多くのタイムアウトと再送を取得しており、さらにサーバーがアイドル状態であることが認識されている場合は、パケットがネットワーク上で破棄されていると考えられます。

ここで言うネットワーク とは、ルーター、ブリッジ、コンセントレーターなどのメディアやネットワーク・デバイスを含む非常にさまざまな種類のもの、および、クライアントとサーバー間のパケットのトランスポートをインプリメントできるようにするすべての範囲の事物を指します。

サーバーが過負荷でなく、パケットを破棄していないときに、NFS のパフォーマンスが低い場合は、ネットワーク上でパケットが破棄されていると想定してください。これを証明し、ネットワークがどのようにパケットを破棄しているかを突き止めるには、多くの労力を要します。 問題を判別する最も簡単な方法は、大部分、関係している物理的な近接度と使用可能なリソースによって決まります。

サーバーとクライアントが直接接続できるほど近くにあり、問題を起こす可能性のある大きいネットワークの部分をバイパスできる場合もあります。 当然のことながら、このことを行って、問題が解消される場合は、マシンそのものを問題の原因として除去します。 ただし、直接接続は配線的に不可能な場合が多いので、問題を的確にトレースする必要があります。ネットワークの探知機能およびその他のツールを使用して、このような問題をデバッグしてください。

NFS パフォーマンスのためのディスク・サブシステム構成

読み取り/書き取り集中ワークロードの最も一般的なボトルネックのソースの 1 つは、不適切に構成されたディスク・サブシステムです。

NFS サーバー上のディスク・サブシステムのチューニングだけを検討している場合もありますが、いくつかのシナリオでは、NFS クライアント上で不適切に構成されたディスク・サブシステムのセットアップが実際の問題である場合もあることに注意してください。 その一例は、 NFS クライアント上のアプリケーションが、 NFS マウントされたファイルシステムからクライアント上のローカル・ファイルシステムにファイルをコピーするときのワークロードです。 このケースでは、クライアント上のディスク・サブシステムを適切にチューニングして、ローカル・ファイルシステムへの書き込みパフォーマンスがボトルネックにならないようにすることが重要です。 193 ページの『論理ボリュームおよびディスク入出力のパフォーマンス』に記述されているチューニング・テクニックを参照してください。 特に、次に挙げることを検討してください。

- NFS 上の単純な読み取りまたは書き込みのワークロードについては、使用中のファイルシステムを含んだディスクのパフォーマンス機能を評価してください。 そのためには、ファイルシステム上のファイルに対する書き込みまたは読み取りをローカルに実行します。 テスト・アプリケーションは、実際にディスクにデータを書き込むことなく終了することが多いので、ディスクのスループット機能を測定するために **iostat** コマンドを使用してください。 例えば、メモリー内にデータが残っている可能性もあります。 このローカルな読み取り/書き込みに関して測定したスループットは、追加処理や NFS に関連する待ち時間のオーバーヘッドを伴わないので、NFS 上で達成できる上限のパフォーマンスと見なせる場合が一般的です。
- データ・アクセス時に高い並列性を達成することが必要なときがよくあります。 複数のクライアントまたは複数のクライアント・プロセスが、サーバー上の 1 つのファイルシステムに同時にアクセスすると、特定のデバイスのディスク入出力でスループットがボトルネックとなることがあります。 **iostat** コマンドを使用すれば、ディスクのロードを評価することができます。 特に、 *%tm_act* パラメーターは、特定のディスクがアクティブである時間の比率を表しますが、この値が高ければ、関連するディスク・アダプターが過負荷になっている可能性があります。
- ディスク・サブシステムのチューニングには直接関係しませんが、単一ファイルへの同時書き込みは、ファイルの inode ロックの競合という結果になり得るということもここで述べておきます。 ほとんどのファイルシステムは、ファイルへのアクセスをシリアライズするために inode ロックを使用します。これによって、書き込まれるデータの一貫性が確保されます。 ところが、このように複数のスレッドが同じファイルに同時に書き込もうとするケースでは、どの時点でも inode ロックを保持するスレッドにのみファイルへの書き込みが許可されるので、書き込みのパフォーマンスが非常に悪化する可能性があります。
- 大規模 NFS サーバーの場合は、一般的な戦略として、可能な限り多くのディスクとディスク・アダプター・デバイスにディスク入出力要求を均等に分ける方法を用いるべきです。ディスク入出力が適切に分散されているシステムでは、サーバー上の CPU ロードがワークロードのパフォーマンス上の制限要因になる可能性があります。

パフォーマンスに影響する NFS の誤用

アクセスするファイルがコストのかかる通信パスのもう一方の端にあることをユーザーが認識していない場合に、 NFS の誤用が数多く生じます。

以下にその例をいくつか挙げます。

- NFS マウントの在庫ファイルの更新をランダムに行っている 1 つのシステムで実行中のアプリケーション。リアルタイムの小売キャッシュ・レジスターのアプリケーションをサポートする。
- 各システム上のソース・コード・ディレクトリーが、同じ環境内のほかのすべてのシステムに NFS によってマウントされたものである開発環境。その開発者は、任意のシステムにログオンして、編集とコンパイルを行う。 これにより、すべてのコンパイルがそのソース・コードをリモート・システムから取得し、その出力をリモート・システムに書き込むことが実際上保証されています。
- 1 つのシステム上で **ld** コマンドを実行して、NFS によってマウントされたディレクトリー内の `.o` ファイルを、同じディレクトリー内の `a.out` ファイルに変換する。
- ページ位置合わせが行われない書き込みを発行するアプリケーション (例えば 10 KB)。 サイズが 4 KB より小さい書き込みは、常にページイン を発生させ、NFS の場合は、このページイン がネットワーク上に送られます。

これらは、NFS が提供する透過性の有効な使用法であると主張できるかもしれません。 おそらくそうであるかもしれませんが、これらの使用法ではプロセッサー時間と LAN の帯域幅が費やされるので、応答時間は遅くなります。 あるシステム構成にオペレーションの標準パターンの一部として NFS アクセスが含まれている場合には、構成の設計者は、結果として生じるコストを、次のような技術的あるいはビジネス上の利点によって正当化できる用意がなければなりません。

- すべてのデータまたはソース・コードを個々のワークステーションではなく、サーバーに入れると、ソース・コードの制御が改善され、集中化されたバックアップが単純化されます。
- いくつかの異なるシステムが同じデータにアクセスすることによって、クライアントとサーバーの役割を兼備している 1 つ以上のシステムと比べて、専用サーバーをより効率的にします。

NFS ファイルシステム上で実行すべきでないもう 1 つのタイプのアプリケーションは、毎秒数百の **lockf()** または **flock()** コールを実行するアプリケーションです。 NFS ファイルシステムでは、すべての **lockf()** または **flock()** コール (およびその他のファイル・ロック・コール) が、 **rpc.lockd** デーモンを介して処理される必要があります。これにより、ロック・デーモンが毎秒数千のロック要求を処理することはできないため、システム・パフォーマンスが大幅に低下することがあります。

クライアントとサーバーのパフォーマンスの能力には無関係に、NFS ファイル・ロックに関連するすべてのオペレーションが、不当に低速になったと思われることがあります。 この技術的な理由は幾つかありますが、ファイルがロックされている場合は、そのファイルが読み取りと書き込みの両方の側で同期処理されるように、特別な配慮が必要であることに起因しています。 これは、クライアントには、ファイル属性を含むあらゆるファイル・データのキャッシングが存在し得ないということを意味します。 すべてのファイル・オペレーションは、キャッシングなしの完全同期モードになります。アプリケーションが NFS 上で作動しており、同じクライアント/サーバーの対上にあるほかのアプリケーションに比べて異常にパフォーマンスが低いときは、そのアプリケーションがネットワーク・ファイル・ロックを実行しているのではないかと疑う必要があります。

サーバー上の NFS パフォーマンス・モニター

サーバーのプロセッサー、メモリー、および入出力の構成が適切かどうかを確認するために、ワークロード・アクティビティー実行時の CPU 使用率、入出力アクティビティー、およびメモリー使用率をチェックするには、NFS サーバー上で **vmstat** および **iostat** コマンドを使用します。

nfsstat コマンドを使用して、サーバー上の NFS オペレーション・アクティビティーをモニターできます。

nfsstat -s コマンド

NFS サーバーは、受け取った NFS コールの数 (「calls」) と、認証によってリジェクトされた NFS コールの数 (「badcalls」) のほかに、発行された各種コールのカウントとパーセントを表示します。

以下の例には、**-s** オプションで指定された **nfsstat** コマンド出力のサーバー部分が示されています。

```
# nfsstat -s

Server rpc:
Connection oriented:
calls     badcalls  nullrecv badlen    xdrcall  dupchecks dupreqs
15835      0          0          0          0          772        0
Connectionless:
calls     badcalls  nullrecv badlen    xdrcall  dupchecks dupreqs
0          0          0          0          0          0          0

Server nfs:
calls      badcalls   public_v2  public_v3
15835      0          0          0
Version 2: (0 calls)
null       getattr    setattr    root       lookup     readlink   read
0 0%       0 0%       0 0%       0 0%       0 0%       0 0%       0 0%
wrcache    write      create     remove     rename     link       symlink
0 0%       0 0%       0 0%       0 0%       0 0%       0 0%       0 0%
mkdir      rmdir      readdir    statfs
0 0%       0 0%       0 0%       0 0%
Version 3: (15835 calls)
null       getattr    setattr    lookup     access     readlink   read
7 0%       3033 19%   55 0%      1008 6%    1542 9%    20 0%      9000 56%
write      create     mkdir      symlink    mknod      remove     rmdir
175 1%     185 1%     0 0%       0 0%       0 0%       120 0%     0 0%
rename     link       readdir    readdir+   fsstat     fsinfo     pathconf
87 0%      0 0%       1 0%       150 0%     348 2%     7 0%       0 0%
commit
97 0%
```

サーバーの RPC 出力 **-s** は次のとおりです。

calls クライアントから受け取った RPC コールの合計数

badcalls
RPC レイヤーがリジェクトしたコールの合計数

nullrecv
RPC コールが受信されたはずなのに入手できなかった回数

badlen
切り捨てられたかまたは損傷したパケットの数 (最小サイズの RPC コールより短い長さの RPC コールの数)

xdrcall
ヘッダーを外部データ表現 (XDR) でデコードできなかった RPC コールの数

dupchecks
重複要求キャッシュで検索された RPC コールの数

dupreqs
検出された重複 RPC コールの数

この出力には、各種コールのカウントと、それぞれのパーセントも表示されます。

二度実行して同じ結果が得られることのないようなオペレーションについては、重複に関する検査が行われます。 典型的な例として、**rm** コマンドがあります。最初の **rm** コマンドは成功しますが、応答が失われると、クライアントはそのコマンドを再送します。 このような重複している要求を成功させるため、重複するキャッシュが調べられ、要求が重複しているときには、重複している要求に対して、最初の要求時に生成されたのと同じ (成功の) 結果が戻されます。

各種オペレーション、例えば、 **getattr()**、**read()**、**write()**、または **readdir()** などのコールのパーセントを調べることによって、使用するチューニングのタイプを判断できます。 例えば、**getattr()** コールのパーセントが非常に高い場合は、属性キャッシュをチューニングするとよいでしょう。 **write()** コールのパーセントが非常に高い場合は、ディスクと LVM のチューニングが重要です。 **read()** コールのパーセントが非常に高い場合は、ファイルのキャッシングのためにより多くの RAM を使用すると、パフォーマンスが向上することがあります。

サーバー上の NFS パフォーマンス・チューニング

サーバー上の NFS 特有のチューニング変数は、主に **nfso** コマンドを使用してアクセスすることが可能です。

一般的に、NFS 特有オプションのチューニングを適切にインプリメントすれば、以下に示すような問題の解決に役立ちます。

- ネットワークおよび NFS サーバー上の負荷の低減
- ネットワーク上の問題やクライアントのメモリー使用率の問題への対処

必要な nfsd スレッドの数

NFS サーバー上には、マルチスレッドの 1 つの **nfsd** デーモンが存在します。 これは、**nfsd** プロセスに、複数のカーネル・スレッドが存在することを意味します。 また、スレッド数は自己チューニング型です。つまり、デーモンは NFS の負荷に応じてスレッドを作成および破棄します。

この自己チューニング機能があるために、また最大 **nfsd** スレッド数のデフォルト値 (3891) がいずれにしても許可されている最大値であることからすれば、この値を変更する必要はまずありません。 ただし、システムの最大 **nfsd** スレッド数は、 **nfso** コマンドの *nfs_max_threads* パラメーターを使用して調整できます。

サーバー上の読み取りおよび書き込みサイズの制限

nfso コマンドの **nfs_max_read_size** および **nfs_max_write_size** オプションを使用して、 NFS 読み込み応答および NFS 書き込み要求に使用される RPC の最大サイズをそれぞれ制御できます。

376 ページの『クライアント上の NFS チューニング』のセクションでは、読み取りおよび書き込み RPC サイズをチューニングするための適切な状況について説明しています。 基本的に、チューニングを実行するのはクライアント上です。 しかし、クライアント上のこれらの値の変更を管理するのが難しい環境では、サーバーの **nfso** オプションが便利です。

ファイル・データ・チューニングの最大キャッシュ

NFS は、NFS エクスポート・ファイルシステム内にファイル・データをキャッシュするための自身専用のバッファーを持ちません。

代わりに仮想メモリー・マネージャー (VMM) が、これらのファイル・ページのキャッシュを制御します。 システムが専用 NFS サーバーとして動作する場合は、データ・キャッシュに必要な分のメモリーを

使用するように VMM に許可することが適切です。 JFS ファイルシステムをエクスポートするサーバーの場合、これを行うには、*maxperm* パラメーターを設定します。このパラメーターは、100% までの JFS ファイル・ページが占有するメモリーの最大パーセンテージを制御します。 このパラメーターを設定するには、 **vmo** コマンドを使用します。 次に例を示します。

```
# vmo –o maxperm%=100
```

拡張 JFS ファイルシステムをエクスポートするサーバーでは、*maxclient* および *maxperm* パラメーターの両方が設定されている必要があります。 *maxclient* パラメーターは、拡張 JFS ファイル・データがキャッシュされる、クライアント・セグメント・ページが占有するメモリーの最大パーセンテージを制御します。 *maxclient* の値は *maxperm* の値を超えられないことに注意してください。 次に例を示します。

```
# vmo –o maxclient%=100
```

特定の条件下では、メモリー内にキャッシュするファイル・データが多すぎることは、実際に好ましくありません。 アプリケーションで再利用しないと思われるファイル・データをフラッシュするために、*release-behind* と呼ばれる機構をどのように使用するかの説明については、 254 ページの『ファイルシステムのパフォーマンス』を参照してください。

RPC マウント・デーモンのチューニング

rpc.mountd デーモンはマルチスレッド化されており、デフォルトで最高 16 スレッドまで作成できます。

automount デーモンがよく使用される環境で、**automount** デーモンのタイムアウトが頻繁に見られる場合は、最大 **rpc.mountd** スレッドの数を以下のように増やす必要があります。

```
# chsys -s rpc.mountd -a –h <number of threads>
# stopsrc -s rpc.mountd
# startsrc -s rpc.mountd
```

RPC ロック・デーモンのチューニング

rpc.lockd デーモンはマルチスレッド化されており、デフォルトで最高 33 スレッドまで作成できます。

RPC ファイル・ロック・アクティビティーが激しい場合は、いったん最大スレッド数に到達すると、**rpc.lockd** デーモンがボトルネックになる可能性があります。 最大値に到達すると、後続の要求は待機しなければならなくなり、その結果、他のタイムアウトになる場合があります。 **rpc.lockd** スレッド数を最大で 511 に調整できます。 次に例を示します。

```
# chsys -s rpc.lockd -a <number of threads>
# stopsrc -s rpc.lockd
# startsrc -s rpc.lockd
```

クライアント上の NFS パフォーマンス・モニター

クライアントのプロセッサーおよびメモリーの構成が適切かどうかを確認するために、ワークロード・アクティビティー実行時の CPU 使用率およびメモリー使用率をチェックするには、NFS クライアント上で **vmstat** コマンドを使用します。

nfsstat コマンドを使用して、クライアントによる NFS オペレーション・アクティビティーをモニターできます。

nfsstat -c コマンド

NFS クライアントは、送信およびリジェクトされるコールの数を表示するのと同時に、クライアント・ハンドルが受け取られた回数 `clgets` および各種コールのカウントとそれぞれのパーセントを表示します。

以下の例は、**-c** オプションを使用してクライアントに指定された **nfsstat** コマンドの出力です。

```
# nfsstat -c

Client rpc:
Connection oriented
calls     badcalls  badxids  timeouts  newcreds badverfs  timers
0          0          0          0          0          0          0
nomem     cantconn interrupts
0          0          0
Connectionless
calls     badcalls retrans   badxids   timeouts newcreds  badverfs
6553       0          0          0          0          0          0
timers    nomem     cantsend
0          0          0


Client nfs:
calls      badcalls   clgets     cltoomany
6541       0          0          0
Version 2: (6541 calls)
null       getattr    setattr    root       lookup     readlink   read
0 0%       590 9%     414 6%     0 0%       2308 35%   0 0%       0 0%
wrcache    write      create     remove     rename     link       symlink
0 0%       2482 37%   276 4%     277 4%     147 2%     0 0%       0 0%
mkdir      rmdir      readdir    statfs
6 0%       6 0%       30 0%      5 0%
Version 3: (0 calls)
null       getattr    setattr    lookup     access     readlink   read
0 0%       0 0%       0 0%       0 0%       0 0%       0 0%       0 0%
write      create     mkdir      symlink    mknod      remove     rmdir
0 0%       0 0%       0 0%       0 0%       0 0%       0 0%       0 0%
rename     link       readdir    readdir+   fsstat     fsinfo     pathconf
0 0%       0 0%       0 0%       0 0%       0 0%       0 0%       0 0%
commit
0 0%
```

クライアントの RPC 出力 **-c** は次のとおりです。

calls NFS に対して行われた RPC コールの合計数

badcalls
RPC レイヤーがリジェクトしたコールの合計数

retrans
サーバーからの応答を待っているときのタイムアウトが原因でコールを再送する必要があった回数。 これは、コネクションレス・トランスポートを用いる RPC だけに適用されます。

badxid
未解決のコールに対応しなかった応答をサーバーから受け取った回数。 これは、サーバーが応答するまでに時間がかかりすぎているということを意味します。

timeouts
サーバーからの応答を待っているときにコールがタイムアウトになった回数。

newcreds
認証情報を更新する必要があった回数。

badverfs
応答内の無効なベリファイヤーが原因でコールが失敗した回数。

timers
算出されたタイムアウト値がコールに指定された最小タイムアウト値以上だった回数。

nomem
メモリーを割り当てることができずにコールが失敗した回数。

cantconn
サーバーへの接続ができずにコールが失敗した回数。

interrupts
コールが、完了する前にシグナルによって割り込まれた回数。

cantsend
クライアントへの接続ができずに送信が失敗した回数。

この出力には、各種コールのカウントと、それぞれのパーセントも表示されます。

パフォーマンスをモニターするために、**nfsstat -c** コマンドは、ネットワークが UDP パケットを破棄しているかどうかについての情報を提供します。 ネットワークは、パケットを処理できないときに、破棄することがあります。 また、ネットワーク・ハードウェアまたはソフトウェアの応答時間、あるいはサーバー上の CPU の過負荷が原因となって、パケットが破棄されることもあります。 破棄されたパケットは、それに対して置換要求が発行されるので、実際には失われません。

RPC セクションの「*retrans*」欄には、応答を待っているときのタイムアウトが原因で要求が再送された回数が表示されます。この状況は、破棄された UDP パケットに関係があります。 「*retrans*」値が一貫して欄 1 の合計コール数の 5% を超える場合は、サーバーが要求の速さについていけるかどうかが問題となっていることが示されています。 サーバー・マシン上で **vmstat**、および **iostat** コマンドを使用して負荷を調べてください。

「*badxid*」のカウントが大きい場合は、要求が各種の NFS サーバーに達してはいるが、サーバーの負荷が多きすぎるために、クライアントの RPC コールがタイムアウトになって再送される前に、サーバーが応答を送信できなくなっています。 「*badxid*」値は、伝送された要求に対して重複する応答を受け取るたびに増分されます。 RPC 要求はすべての送信サイクルでその「*XID*」値を保持しています。 過度の再送は、サーバーに過度の負担をかける結果となり、応答時間がさらに遅くなります。 「*badxid*」値とタイムアウト数が、合計コール数の 5% を超える場合は、**smitty chnfsmnt** コマンドを使用して、 NFS マウント・オプションの *timeo* パラメーターを大きくしてください。 「*badxid*」値は 0 だが、「*retrans*」値とタイムアウト数が相当多い場合は、**mount** コマンドの **rsize** および **wsize** オプションを使用して、NFS バッファー・サイズを小さくしてみてください。

再送数とタイムアウトの数が近い値である場合は、確実にパケットが破棄されています。 詳細については、 367 ページの『破棄されたパケット』を参照してください。

場合によっては、アプリケーションまたはユーザーがパフォーマンスの低下を経験しているのに、**nfsstat -c** コマンドの出力を調べると、タイムアウト数と再送数が非常に少ないかまたは 0 を示していることがあります。 これは、クライアントが応答を求めるとすぐに、サーバーから応答を受け取っていることを意味します。 最初に調べるべきことは、クライアント・マシン上で実行中の **biod** デーモンが適切な数だけ存在するかどうかです。 これは、アプリケーションがリモート・ファイル・ロックを行っているときにも、監視することができます。 リモート・ファイル・ロックが NFS でサービスを受けるファイルに設定されると、クライアントは完全に同期モードのオペレーションに入り、そのオペレーションによってファイルのデータと属性すべてのキャッシングがオフになります。 その結果、パフォーマンスが非常に低下しますが、これが残念ながら通常の状況です。 パケットのロックは、**ipreport** の出力で NLM 要求を調べることによって識別できます。

nfsstat -m コマンド

nfsstat -m コマンドは、サーバーの名前およびアドレス、マウント・フラグ、現在の読み取りサイズおよび書き込みサイズ、再送数、およびクライアント上の NFS マウントごとに動的再送に使用されるタイマーを表示します。

次に例を示します。

```
# nfsstat -m
/SAVE from /SAVE:aixhost.ibm.com
 Flags:   vers=2,proto=udp,auth=unix,soft,intr,dynamic,rsize=8192,wsize=8192,retrans=5
 Lookups: srtt=27 (67ms), dev=17 (85ms), cur=11 (220ms)
 Reads:   srtt=16 (40ms), dev=7 (35ms), cur=5 (100ms)
 Writes:  srtt=42 (105ms), dev=14 (70ms), cur=12 (240ms)
 All:     srtt=27 (67ms), dev=17 (85ms), cur=11 (220ms)
```

出力例の括弧内の数は、ミリ秒単位の実際の時間です。 その他の値は、オペレーティング・システムのカーネルによって保持されているスケーリングされていない値です。 スケーリングされていない値は無視できます。 応答時間は、ルックアップ、読み取り、書き込み、およびこれらのすべてのオペレーションの組み合わせ (「All」) について示されます。この出力に使用されているその他の定義は、次のとおりです。

srtt 平滑化された往復時間

dev 偏差の見積もり

cur 現在のバックオフされたタイムアウト値

クライアント上の NFS チューニング

NFS 特有のチューニング変数は、基本的に **nfso** および **mount** コマンドを使用してアクセスすることが可能です。

チューニング変数の調整を開始する前に、これらの値を変更することで達成すべき目標は何か、これらの変更によって生じ得るマイナスの副次作用は何かについて明確に理解しておいてください。

/etc/filesystems スタンザを変更することによって、特定のファイルシステムの **mount** オプションを設定できます。これにより、ブート時のファイルシステムのマウントの際にこれらの値が有効になります。

一般的に、NFS 特有オプションのチューニングを適切にインプリメントすれば、以下に示すような問題の解決に役立ちます。

- ネットワークおよび NFS サーバー上の負荷の低減
- ネットワーク上の問題やクライアントのメモリー使用率の問題への対処

必要な biod スレッドの数

NFS クライアント上には、マルチスレッドの 1 つの **biod** デーモンが存在します。 これは、**biod** プロセスに、複数のカーネル・スレッドが存在することを意味します。 また、スレッド数は自己チューニング型です。つまり、デーモンは NFS の負荷に応じてスレッドを作成および破棄します。

また、**biod** マウント・オプションを使用して、マウントごとに **biod** スレッドの最大数をチューニングすることもできます。 **biod** スレッド数のデフォルトは、NFS バージョン 3 マウントおよび NFS バージョン 4 マウントでは 4、 NFS バージョン 2 マウントでは 7 です。

biod スレッドは、一度に 1 つの読み取りまたは書き込み要求を処理し、NFS の応答時間は多くの場合、応答時間全体の中で一番大きい部分を構成しているので、**biod** スレッドが不足しているためにアプリケーションがブロックされるのは望ましくありません。

nfsd および **biod** デーモンの最適な数を判別するのは、反復プロセスです。 以下に示すガイドラインは、単に合理的な開始点にすぎません。 **biod** スレッドの構成に関する一般的な考慮事項は、以下のとおりです。

- スレッドの数を増やしても、不十分なクライアントまたはサーバー・プロセッサーの能力やメモリー、あるいは不十分なサーバー・ディスクの処理能力を補償することはできません。スレッドの数を変更する前に、**iostat** および **vmstat** コマンドを使用して、サーバーとクライアントのリソース使用率レベルを調べてください。
- CPU またはディスク・サブシステムが既に飽和状態に近づいている場合は、スレッドの数を増やしても、パフォーマンスは向上しません。
- 読み取りと書き込みだけが、**biod** スレッドを介して送られます。
- このデフォルトが一般的な開始点となりますが、複数のアプリケーション・スレッドがマウント・ポイント上のファイルに同時にアクセスする場合は、マウント・ポイントの **biod** スレッドの数を増やすことが望ましい場合があります。 例えば、同時に書き込みが行われるファイル数を見積もりたい場合があります。先読みと遅延書き込みのアクティビティーをサポートするために、1 つのファイルに対して最低でも 2 つの **biod** スレッドが存在することを確認してください。
- 高速のクライアント・ワークステーションが低速のサーバーに接続されているときには、クライアントが NFS 要求を生成する速度を制限する必要がある場合があります。考えられる解決策は、各クライアントのワークロードと応答時間の相対的な重要性に注意しながら、クライアント上の **biod** スレッドの数を減らすことです。 クライアント上の **biod** スレッドの数を大きくすると、クライアントが一度に送信できる要求が増えて、ネットワークとサーバーにかかる負荷が大きくなるので、サーバーのパフォーマンスに悪い影響を与えます。 クライアントがサーバーをオーバーランする場合には、**biod** スレッドの数を 1 まで減らす必要があるかもしれません。 次に例を示します。

  ```
  # stopsrc -s biod
  ```

  上記の例では、クライアントは **biod** カーネル・プロセスをそのまま実行している状態になります。

読み取りおよび書き込みサイズの調整

最も役に立つ NFS チューニング・オプションは、**rsize** および **wsize** オプションです。これらのオプションを使って、読み取りおよび書き込みの RPC パケットの最大サイズをそれぞれ定義できます。

読み取りおよび書き込みのサイズ値を変更する理由の概略を以下に示します。

- サーバーは、読み取り/書き込みパケットの転送に必要なデータ・ボリュームと速度 (NFS バージョン 2 の場合は 8 KB、NFS バージョン 3 および NFS バージョン 4 の場合は 32 KB) を扱えないことがあります。NFS クライアントが NFS サーバーとして PC を使用している場合がこのケースにあたります。 PC では、大きなパケットのバッファリングに使用できるメモリーが制約されていることがあります。
- 読み取り/書き込みサイズ値を小さくすると、後で、コールが生成する IP フラグメントの数が減少することがあります。 障害のあるネットワークを扱っているときには、 7 個のパケットが正常に交換される必要があるのに、 2 個のパケット交換だけでコールおよび応答の対が完了する場合が多くなります。同様に、パフォーマンス特性が異なる複数のネットワークにわたって NFS パケットを送信する場合は、 IP フラグメントのタイムアウト値より前に、すべてのパケット・フラグメントが到着しないことがあります。

混雑したネットワークの場合は、**rsize** および **wsize** を小さくすると、各 NFS 読み取り応答および書き込み要求ごとに送られるパケットが短いので、NFS パフォーマンスは向上することがあります。 しかし、ネットワーク上にデータを送信するのにより多くのパケットが必要になり、ネットワーク・トラフィックの合計量が増えるのと同時に、サーバーとクライアントの両方の CPU 使用率が増えるという副次作用が生じます。

NFS ファイルシステムがギガビット・イーサネットなどの高速ネットワークにマウントされると、読み取りおよび書き込みパケット・サイズが大きくなるので、NFS ファイルシステムのパフォーマンスが向上し

ます。 NFS バージョン 3 および NFS バージョン 4 で、ネットワーク・トランスポートが TCP の場合は、 **rsize** と **wsize** の値を最高で 65536 に設定できます。 デフォルト値は 32768 です。 NFS バージョン 2 では、**rsize** および **wsize** オプションの最大値は 8192 で、これがデフォルトです。

NFS ファイル・データのキャッシングのチューニング

VMM は、NFS クライアント上のクライアント・セグメント・ページ内の NFS ファイル・データのキャッシュを制御します。

NFS クライアントがワークロードを実行中で、少量の作業セグメント・ページを必要としている場合、VMM にできるだけ多くのシステム・メモリーを NFS ファイル・データ・キャッシュとして使用させることが適切な場合があります。 そのためには、*maxperm* パラメーターと *maxclient* パラメーターの両方を設定します。 *maxclient* の値は、 *maxperm* の値以下でなければなりません。 以下は、ファイル・キャッシュとして使用可能なメモリーの量を 100% に設定する例です。

```
# vmo –o maxperm%=100
# vmo –o maxclient%=100
```

読み取りスループットに対する **NFS** データ・キャッシュの影響**:**

クライアントで測定される NFS の順次読み取りスループットは、VMM の先読みとキャッシュのメカニズムによって拡張されます。

先読みにより、ファイル・データは、NFS クライアント・アプリケーションが要求することを予期して、NFS サーバーからクライアントへ転送されます。データに対する要求がアプリケーションによって発行されるまでに、データは既にクライアントのメモリー内にある可能性があるので、要求を即時に満たすことができることがあります。 VMM キャッシュでは、クライアント・メモリーからデータがページアウトされていないことを前提に (ページアウトされている場合は、データを NFS サーバーからもう一度取り出す必要がある)、ファイル・データの再読み取りが即時に行えます。

クライアント上での NFS データの VMM キャッシングにより利点を得るアプリケーションは多くありますが、データベースなど、一部のアプリケーションは、それぞれ独自のファイル・データ・キャッシュ管理を行います。 これらの独自のファイル・データ・キャッシュ管理を行うアプリケーションは、NFS を介して直接 I/O、または DIO を使用することにより利点を得ることができます。 **mount** コマンドの **dio** オプションを使用して、または **open()** システム呼び出しで **O_DIRECT** フラグを指定することによって、NFS を介した DIO を使用可能にすることができます。

以下のリストに、DIO の利点を詳しく示します。

- VMM とアプリケーションがファイル・データを二重にキャッシングしないようにします。
- DIO 機能は VMM コードをバイパスするため、ファイルの読み取りおよび書き込みでの CPU 効率が向上します。

データベースなど、独自のファイル・データ・キャッシュ管理およびファイル・アクセス・シリアライズを行うアプリケーションは、並行 I/O、または CIO を使用することにより利点を得ます。 DIO の利点に加えて、CIO は、ファイル・アクセスの読み取りおよび書き込みをシリアライズしないため、マルチスレッドが同一ファイルに対して読み込みまたは書き込みを同時に行えます。

注**:** CIO または DIO を使用すると、一部のアプリケーションのパフォーマンスが低下する可能性があります。それは、VMM ファイル・キャッシングに大きく依存するアプリケーション、およびシステム・パフォーマンスを増大させるための先読みおよび後書き VMM 最適化に大きく依存するアプリケーションです。

CacheFS を使用して次のような環境 (メモリー制約のクライアント、非常に大きいファイル、低速のネットワーク・セグメントのある環境のすべてまたはいずれか) での読み取りスループットをさらに拡張可能です。これを行うには、クライアント上のローカル・ディスク・キャッシュ内にあるファイル・データからの読み取り要求を満足させる潜在能力を追加します。 詳しくは、 382 ページの『キャッシュ・ファイルシステム』を参照してください。

大規模ファイルを順次に読み取るためのデータ・キャッシングは、メモリーが NFS データ・キャッシュで一杯になるため、ページ置換アクティビティーが頻繁に行われる可能性があります。 release-behind on read、**rbr**、**mount** オプションまたは NFS バージョン 4 の **nfs4cl setfsoptions** 引数を使用してページ置換アクティビティーを回避することによってパフォーマンスを向上させることができます。ラージ・ファイルの順次読み取りの場合、以前の読み取りデータが入った実メモリーが、順次読み取りの続行に合わせて解放されます。

rbr mount オプションが、すぐに再度必要になるメモリー解放を開始する場合、代わりに、**nfso** コマンドの **nfs_auto_rbr_trigger** チューニング・オプションを使用できます。 **nfs_auto_rbr_trigger** チューニング・オプション (メガバイト単位で測定) は、 release-behind の読み取り時オプションが有効になると、読み取りオフセットしきい値として機能します。 例えば、**nfs_auto_rbr_trigger** チューニング・オプションが 100 MB に設定されている場合、順次に読み取られるファイルの最初の 100 MB はキャッシュに入れられ、ファイルの残りはメモリーから解放されます。

書き込みスループットに対する **NFS** データ・キャッシュの影響**:**

クライアント・メモリーより大きなファイルの順次書き込み操作を実施しようとする場合、NFS バージョン 3 または NFS バージョン 4 では commit-behind を使用することによってパフォーマンスを改善できます。

クライアントでメモリー量よりも大きなファイルの全体を書き込む場合には、クライアントで大量のページ置き換えアクティビティーが発生します。 これによって、書き込まれたデータ・ページごとに、ネットワーク経由でコミット・オペレーションが発生することもあります。 commit-behind の場合は、サーバー上の安定したストレージにクライアント・ページをコミットするロジックや、さらに重要なこととして、それらのページをフリー・リストに戻すロジックがもっと積極的です。

combehind オプションを **mount** コマンドに指定することにより、ファイルシステムのマウント時に commit-behind を使用可能にできます。 **mount** コマンドを使用して、*numclust* 変数に適切な値を設定する必要もあります。 この変数は、仮想メモリー管理機能 (VMM) の順次 write-behind アルゴリズムによって処理される、16KB クラスターの数を指定します。 入出力のパターンが順次である場合は、入出力をスケジュールする前に大きな値を **numclust** オプションに使用して、より多くのページが RAM 内に残るようにします。ストライプ論理ボリュームまたはディスク・アレイが使用される場合は、**numclust** オプションの値を大きくしてください。

NFS ファイル属性キャッシュのチューニング

NFS は、最近アクセスされたディレクトリーとファイルに関する属性のキャッシュを各クライアント・システム上に維持します。

mount で設定されるいくつかのパラメーターは、任意のエントリーがキャッシュに保持される時間を制御します。 それらのパラメーターを以下に示します。

actimeo

ファイルおよびディレクトリー・エントリーが、更新後にファイル属性キャッシュに保持される絶対時間。 この値は、指定されると、それに続く **min* および **max* 値をオーバーライドして、それらを実際に **actimeo** 値に設定します。

acregmin
更新後にファイル・エントリーが保持される最小時間。 デフォルトは 3 秒です。

acregmax
更新後にファイル・エントリーが保持される最大時間。 デフォルトは 60 秒です。

acdirmin
更新後にディレクトリー・エントリーが保持される最小時間。デフォルトは 30 秒です。

acdirmax
更新後にディレクトリー・エントリーが保持される最大時間。デフォルトは 60 秒です。

ファイルまたはディレクトリーが更新されるたびに、その除去は少なくとも **acregmin** または **acdirmin** 秒間、延期されます。 2 回目以降の更新では、エントリーが少なくとも最後の 2 回の更新間の間隔だけ保持されますが、**acregmax** または **acdirmax** 秒間より長く保持されることはありません。

ハードまたはソフト NFS マウントがパフォーマンスに対して持つ意味

NFS マウント・ディレクトリーの構成時に行う選択の 1 つは、マウントをハード・マウント (**-o hard**) にするか、ソフト・マウント (**-o soft**) にするかです。

マウントが成功したあと、ソフト・マウントされたディレクトリーにアクセスしたときにエラー (一般的には、タイムアウト) が起きると、そのエラーはリモート・アクセスを要求したプログラムにすぐに報告されます。 ハード・マウントされたディレクトリーにアクセスしたときにエラーが起きると、NFS がオペレーションを再試行します。

ハード・マウントされたディレクトリーへアクセスしたときに永続的なエラーが起きると、パフォーマンス上の問題に発展する可能性があります。これは、デフォルトの再試行回数 1000 とデフォルトのタイムアウト値 0.7 秒の組み合わせが、再試行が連続して行われる場合にタイムアウト値を大きくするアルゴリズムと一緒になって、NFS がオペレーションの完了を試行し続けることを意味しているからです。

再試行回数を減らすこと、タイムアウト値を小さくすること、またはその両方を行うことは、**mount** コマンドのオプションを使用すると、技術的には可能です。しかし、パフォーマンス上の問題を取り除くのに十分なだけこれらの値を大きく変更すると、不要なハード・エラーが報告されることがあります。 代わりに **intr** オプションを使用して、ハード・マウント・ディレクトリーをマウントして、ユーザーが、再試行ループに入っているプロセスにキーボードから割り込むことができるようにします。

ディレクトリーをソフト・マウントした場合、エラーはすぐに検出されますが、データ破壊という重大なリスクが伴います。 一般的には、読み取り/書き込みディレクトリーをハード・マウントしてください。

不必要な再送

特定のネットワーク構成のタイムアウト時間を適切な時間に設定するという問題には、ハード・マウント対ソフト・マウントの問題が関連しています。

サーバーが、過負荷の場合、または 1 つ以上のブリッジまたはゲートウェイによってクライアントから離れている場合、あるいは WAN によってクライアントに接続されている場合には、デフォルトのタイムアウトの基準が現実的でないことがあります。 そのような場合には、サーバーとクライアントの両方に、不要な再送による負荷が課されます。 例えば、次のコマンドを実行したとします。

```
# nfsstat -c
```

このコマンドによって、「timeouts」と「badxids」の両方の合計が 5% を超えるようなかなり大きな数値が報告された場合は、 **mount** コマンドを使用して、*timeo* パラメーターを大きくすることができます。

変更したいディレクトリーを識別し、「**NFS TIMEOUT**」の行に新しい値を 10 分の 1 秒単位で入力します。

デフォルトの時間は 0.7 秒 **timeo=7** ですが、この値は、コールのタイプに応じて、NFS カーネル・エクステンション内で操作されます。 例えば読み取りコールの場合は、値が 2 倍の 1.4 秒に設定されます。

オペレーティング・システム バージョン 4 のクライアントの *timeo* 値をうまく制御するには、**nfso** コマンドの **nfs_dynamic_retrans** オプションを 0 に設定する必要があります。 *timeo* 値を変更できる方法についての指示は 2 とおりあり、どのような場合でもどちらか 1 つだけが正しい変更方法です。 タイムアウトを長く、または短くするための正しい方法は、割り当てられた時間内にパケットが到着しない理由によって異なります。

パケットが、単に遅れるだけで、最終的には到着する場合は、*timeo* 変数を長くして、要求が再送される前に応答を戻すことができるようにします。

しかし、パケットが破棄され、クライアントに到着しない場合は、応答を待つ時間が無駄なので、*timeo* を短くします。

どちらを選択するか判断する 1 つの方法として、クライアントの **nfsstat -cr** 出力を調べて、クライアントが報告する「`badxid`」のカウントが大きいかどうかを確認する方法があります。 「`badxid`」値は、RPC クライアントが、予期されていたものと違う別のコール用の RPC コール応答を受け取ったことを意味します。 一般的には、これは、クライアントが、直前に再送されたコールに対する重複する応答を受け取ったということを意味しています。 したがって、パケットは到着が遅れるので、*timeo* を長くする必要があります。

また、ネットワーク・アナライザーを使用できる場合は、それを適用して、2 つの状況のどちらが発生しているかを判断できます。 それ以外の場合は、 *timeo* オプションの設定値を大きくしたり小さくしたりして、どちらがパフォーマンス全体を向上させるかを調べます。 場合によっては、動作に一貫性がないこともあります。そのような場合は、パケット遅延/ドロップの実際の原因を突き止めて、実際の問題 (つまり、サーバーまたはネットワーク/ネットワーク・デバイス) を修正するのが最善策です。

ブリッジを介した 2 つの LAN 間のトラフィックの場合は、10 分の 1 秒単位の値 50 を設定してみてください。 WAN 接続の場合は、値 200 を設定してみてください。 少なくとも 1 日以上たってから、NFS 統計情報をもう一度調べます。 統計情報に依然として過度の再送が示されている場合は、*timeo* 値を 50% 大きくして、再試行してください。 また、サーバーのワークロードと、介在するブリッジおよびゲートウェイの負荷を調べて、ほかのトラフィックによっていずれかのエレメントが飽和状態になっていないか確認します。

未使用の NFS ACL サポート

ワークロードが、マウントされているファイルシステム上の NFS アクセス制御リストまたは ACL サポートを利用しない場合は、**noacl** オプションを指定することによって、クライアントとサーバーの両方のワークロードをある程度減らすことができます。

これは、以下のように行うことができます。

```
options=noacl
```

該当のファイルシステムのクライアントの `/etc/filesystems` スタンザの一部としてこのオプションを設定します。

READDIRPLUS オペレーションの使用法

NFS バージョン 3 では、READDIRPLUS オペレーションによって、ファイル・ハンドルと属性の情報がディレクトリー・エントリーとともに戻されます。 これによってクライアントは、 NFS バージョン 2 の場合とは異なり、その情報をサーバーにエントリーごとに別個に照会する必要がなくなるので、非常に効率的です。

ただし、大きなディレクトリー内の小さいディレクトリー・エントリーのサブセットの情報だけがクライアントによって使用されるような環境では、NFS バージョン 3 の READDIRPLUS オペレーションによって、パフォーマンスが低下することがあります。 このようなケースでは、 **nsfo** コマンドの **nfs_v3_server_readdirplus** オプションを使って、READDIRPLUS を使用不可にすることができます。 ただし、このオプションの使用は NFS バージョン 3 の標準に準拠していないため、通常はお勧めできません。

キャッシュ・ファイルシステム

キャッシュ・ファイルシステムを使用すると、 NFS のようなリモート・ファイルシステムや、CD-ROM などの低速デバイスのパフォーマンスを向上させることができます。

リモート・ファイルシステムをキャッシュに入れると、リモート・ファイルシステムまたは CD-ROM からのデータの読み取りは、ローカル・システム上のキャッシュに保管されるので、同じデータに二度目にアクセスするときに、ネットワークと NFS サーバーを使用する必要がなくなります。 CacheFS は、階層化ファイルシステムとして設計されています。これは、CacheFS が、次の図のように、あるファイルシステム (NFS ファイルシステム、バック・ファイルシステムとも呼ばれる) を別のファイルシステム (ローカル・ファイルシステム、フロント・ファイルシステムとも呼ばれる) のキャッシュに入れる能力を提供していることを意味します。

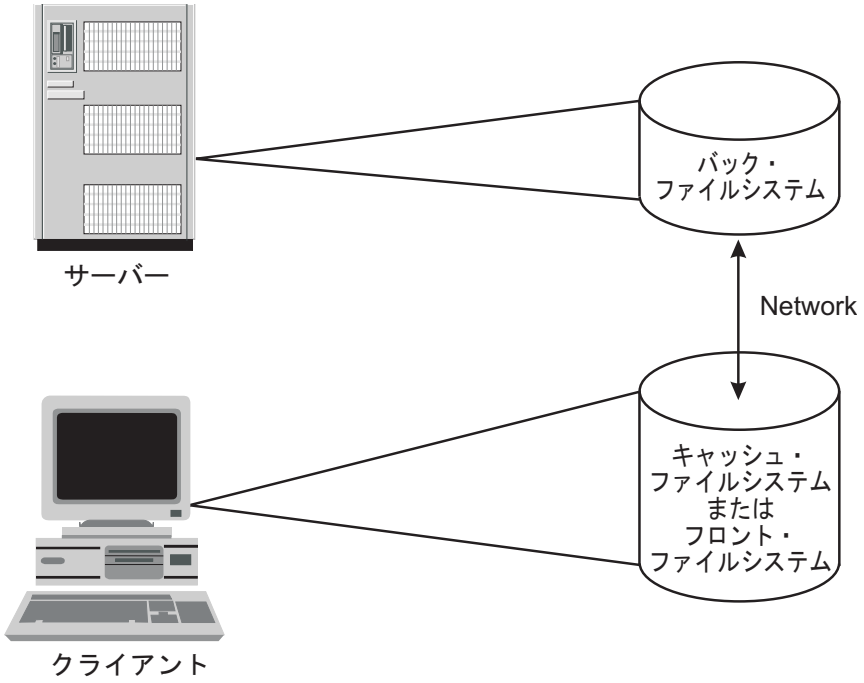


図 *24.* キャッシュ・ファイルシステム *(CacheFS)*： この図は、ネットワークによって接続されたクライアント・マシンとサーバーを示しています。サーバーのストレージ・メディアには、バック・ファイルシステムが入っています。クライアントのストレージ・メディアには、キャッシュ・ファイルシステム (またはフロント・ファイルシステム) が入っています。

CacheFS の機能を以下に示します。

1. クライアント・システムに CacheFS ファイルシステムを作成したならば、キャッシュにマウントするファイルシステムを指定します。
2. クライアント上のユーザーが、バック・ファイルシステムの一部であるファイルにアクセスしようとすると、それらのファイルがキャッシュに入れられます。 ユーザーがファイルへのアクセスを要求するまで、そのキャッシュには何も入れられません。 したがって、ファイルにアクセスする初期要求は通常の NFS 速度ですが、同じファイルへのそれ以降のアクセスはローカルの JFS 速度になります。
3. キャッシュに入れられたディレクトリーとファイルが最新の状態に保たれるようにするため、CacheFS は定期的にキャッシュに保管されたファイルの一貫性を調べます。 これは、現在の変更時刻と直前の変更時刻を比較することによって行います。
4. 変更時刻が異なる場合は、ディレクトリーまたはファイルのすべてのデータと属性が、キャッシュから除去され、新しいデータと属性がバック・ファイルシステムから取り出されます。

CacheFS が適している例としては、描画コンポーネントのマスター・コピーをサーバー上に保持でき、キャッシュに入れられたコピーを使用時にクライアント・ワークステーション上に保持できる CAD 環境があります。

CacheFS では、サイズが 2 GB 以上のファイルからの読み取り、そのようなファイルへの書き込みは認められません。

CacheFS のパフォーマンス上の利点

NFS データは、一度サーバーから読み取られると、ローカル・ディスク上でキャッシュに入れられるので、NFS ファイルシステムに対する読み取り要求は、データをネット上から再び取り出さなければならないときよりも、はるかに速く満たされます。

クライアントのメモリー・サイズと使用率に応じて、少量のデータが保持され、メモリーから取り出されるので、ディスク上のキャッシュ・データの利点は、メモリーに保持できない大量のデータにも適用されます。 別の利点としては、メモリー内でキャッシュに入れられたデータはリブート後に再びサーバーから取り出す必要があるのに対して、ディスク・キャッシュ上のデータはシステムのシャットダウン時にも保持されることが挙げられます。

その他の潜在的な NFS のボトルネックは、低速またはビジー状態のネットワークと、サービスを行う NFS クライアントが多すぎる、作動状況がよくないサーバーです。 したがって、クライアント・システムからサーバーへのアクセスは、低速になる可能性があります。 CacheFS では、ネットワークを通じて最初の読み取りを行いサーバーにアクセスする必要をなくすことはしませんが、同じデータに対するそれ以降の要求では、ネットワークを通じて読み取る必要がありません。

クライアントのローカル・ディスクからより多くの読み取り要求を満たすことができる場合は、サーバーに対する NFS アクセスの量が減少します。 このことは、サーバーがより多くのクライアントにサービスを行えるようになるので、サーバーごとのクライアントの割合が増加することを意味します。

ネットワークを通じての読み取り要求が少なくなると、ネットワークの負荷が減るので、非常にビジーな状態のネットワークまたはその他のデータ転送用のスペースが、ある程度解放されます。

すべてのアプリケーションが CacheFS から利益を得られるわけではありません。 CacheFS は読み取りパフォーマンスの速度を上げるだけなので、主に、同じデータの読み取り要求を大量に実行するアプリケーションが、CacheFS の恩恵を受けます。 計算のために非常に大きいモデルを頻繁にロードする必要があるので、大規模な CAD アプリケーションは確かに CacheFS の恩恵を受けます。

パフォーマンス・テストは、NFS サーバーのメモリーまたはディスクからの読み取りよりも、CacheFS ファイルシステムからの順次読み取りの方が、2.4 から 3.4 倍速いことが分かっています。

CacheFS によるパフォーマンスへの影響

CacheFS によっては、NFS ファイルシステムへの書き込みパフォーマンスは向上しません。ただし、CacheFS をマウントするときには、**mount** コマンドの **-o** オプションのパラメーターとして選択する、いくつかの書き込みオプションがあります。 これらは、それ以降のデータに対する読み取りパフォーマンスに影響します。

書き込みオプションは次のとおりです。

ライト・アラウンド
: ライト・アラウンド・モードはデフォルト・モードで、NFS と同じ方法で書き込み処理を行います。 つまり、バック・ファイルシステムに書き込みが行われ、該当のファイルはキャッシュから除去されます。 このことは、ライト・アラウンドでは、キャッシュが空になるので、書き込み後に、新しいデータをサーバーから取得する必要があることを意味します。

非共有
: キャッシュ・ファイルシステムに書き込むユーザーがほかに存在しないことを確認できる場合は、非共有モードを使用できます。 このモードでは、すべての書き込みがフロントとバックの両方のファイルシステムに行われ、ファイルはキャッシュに入れられたままになります。これは、それ以降の読み取りアクセス時には、サーバーではなく、キャッシュにアクセスするだけで済むことを意味しています。

小さいサイズの読み取りは、いずれにしてもメモリー内に保持される (メモリーの使用率によって異なる) ので、ディスク上のデータをキャッシュに入れても利益はありません。 複数のデータ・ブロックに対するランダム読み取りをキャッシュに入れても、同じデータに繰り返しアクセスするのでなければ、やはり効果はありません。

バック・ファイルシステムの一部であるファイルにユーザーがアクセスしようとするまでに、それらのファイルはキャッシュに入れられるので、初期読み取り要求は、やはりサーバーに送られる必要があります。 初期読み取り要求の場合は、通常の NFS 速度が見られます。 同じデータにそれ以降アクセスするときにだけ、ローカルの JFS アクセス・パフォーマンスを見ることになります。

キャッシュ・データの一貫性は、一定間隔でのみ調べられます。 したがって、頻繁に変更されるデータをキャッシュに入れるのは危険です。 CacheFS は、読み取り専用またはほとんどが読み取りであるデータだけに使用してください。

キャッシュに入れられた NFS ファイルシステムの書き込みパフォーマンスは、NFS バージョン 2 と NFS バージョン 3 とでは異なります。パフォーマンス・テストは、次のことを示しています。

- NFS バージョン 2 を使って新しいファイルを CacheFS マウント・ポイントへ順次書き込みすると、NFS バージョン 2 のマウント・ポイントに直接書き込む場合よりも、速度が 25% 低下します。
- NFS バージョン 3 を使って新しいファイルを CacheFS マウント・ポイントへ順次書き込みすると、NFS バージョン 3 のマウント・ポイントに直接書き込む場合よりも、時間が 6 倍かかります。

CacheFS の構成

CacheFS は、デフォルトではインプリメントされておらず、NFS ファイルシステムの作成時にはプロンプトは出されません。 キャッシュにマウントされるファイルシステムを明示的に指定する必要があります。

以下の方法でファイルシステムをキャッシュにマウントすることを指定します。

1. 次のように、**cfsadmin** コマンドを使用して、ローカル・キャッシュ・ファイルシステムを作成します。

   ```
   # cfsadmin -c -o parameters cache-directory
   ```

   ここで *parameters* はリソース・パラメーターを指定し、 *cache-directory* はキャッシュを作成するディレクトリーの名前です。
2. 次のように、キャッシュにバック・ファイルシステムをマウントします。

   ```
   # mount -V cachefs -o backfstype=nfs,cachedir=/cache-directory remhost:/rem-directory local-mount-point
   ```

   この *rem-directory* はデータがあるリモート・ホストとファイルシステムの名前であり、*local-mount-point* はリモート・ファイルシステムをマウントするクライアント上のマウント・ポイントです。
3. また、SMIT コマンドを使用して、CacheFS を管理することもできます (**smitty cachefs** 高速パスを使用します)。

幾つかのパラメーターは、次のように、作成時に設定できます。

maxblocks
: CacheFS がフロント・ファイルシステム内で要求することが許される最大ブロック数を設定します。 デフォルトは 90% です。

minblocks
: CacheFS がフロント・ファイルシステム内で要求することが許される最小ブロック数を設定します。 デフォルトは 0% です。

threshblocks
CacheFS が *minblocks* で指定されたブロックより多くのブロックを要求できる前に、クライアント側の JFS ファイルシステムで使用可能になっている必要のあるブロック数を設定します。 デフォルトは 85% です。

maxfiles
CacheFS が使用できるファイルの最大数。フロント・ファイルシステム内の i ノードの合計数に対するパーセントで表されます。 デフォルトは 90% です。

minfiles
CacheFS が常時使用を許されているファイルの最小数。フロント・ファイルシステム内の i ノードの合計数に対するパーセントで表されます。 デフォルトは 0% です。

maxfilesize
CacheFS がキャッシュに入れることができるメガバイト単位の最大ファイル・サイズ。 デフォルトは 3 です。

NFS リファレンス

NFS に関連したファイル、コマンド、デーモン、およびサブルーチンが多数あります。

詳細については、ネットワークおよびコミュニケーションの管理およびコマンド・リファレンスを参照してください。

NFS ファイルのリスト

NFS に関連したファイルが多数あります。

以下は、構成情報が含まれている NFS ファイルのリストです。

bootparams
ディスクレス・クライアントがブートに使用できるクライアントがリストされています。

exports
NFS クライアントにエクスポートできるディレクトリーがリストされています。

networks
インターネット・ネットワーク上のネットワークに関する情報が含まれています。

pcnfsd.conf
rpc.pcnfsd デーモンの構成オプションを提供します。

rpc リモート・プロシージャー・コール (RPC) プログラムのデータベース情報が含まれています。

xtab 現在エクスポートされているディレクトリーがリストされています。

/etc/filesystems
システム再始動時にマウントが試行されるすべてのファイルシステムがリストされています。

NFS コマンドのリスト

NFS に関連したコマンドが多数あります。

以下に示すのは、NFS コマンドのリストです。

chnfs 指定された数の **biod** および **nfsd** デーモンを始動します。

mknfs
NFS を実行し、NFS デーモンを始動するように、システムを構成します。

nfso NFS ネットワーク・オプションを構成します。

automount
NFS ファイルシステムを自動的にマウントします。

chnfsexp
NFS によってエクスポートされたディレクトリーの属性を変更します。

chnfsmnt
NFS によってマウントされたディレクトリーの属性を変更します。

exportfs
NFS クライアントへディレクトリーをエクスポートおよびアンエクスポートします。

lsnfsexp
NFS によってエクスポートされたディレクトリーの特性を表示します。

lsnfsmnt
マウントされた NFS システムの特性を表示します。

mknfsexp
NFS を使用してディレクトリーをエクスポートします。

mknfsmnt
NFS を使用してディレクトリーをマウントします。

rmnfs NFS デーモンを停止します。

rmnfsexp
サーバーのエクスポートのリストから NFS によってエクスポートされたディレクトリーを除去します。

rmnfsmnt
クライアントのマウントのリストから NFS によってマウントされたファイルシステムを除去します。

NFS デーモンのリスト

NFS に関連したデーモンが多数あります。

以下に示すのは、NFS ロック・デーモンのリストです。

lockd RPC パッケージを使用してロック要求を処理します。

statd NFS 上のロック・サービスのためのクラッシュとリカバリーの機能を提供します。

以下に示すのは、ネットワーク・サービス・デーモンおよびユーティリティーのリストです。

biod クライアントの読み取りおよび書き込み要求をサーバーへ送信します。

mountd
クライアントからのファイルシステム・マウントの要求に応答します。

nfsd クライアントのファイルシステム・オペレーションについての要求を処理するデーモンを始動します。

pcnfsd
PC-NFS クライアントからのサービス要求を処理します。

nfsstat
マシンがコールを受け取る能力についての情報を表示します。

on リモート・マシン上でコマンドを実行します。

portmap
RPC プログラム番号をインターネット・ポート番号にマップします。

rexd リモート・マシンからのプログラムの実行要求を受け入れます。

rpcgen
RPC プロトコルをインプリメントするための C コードを生成します。

rpcinfo
RPC サーバーの状況を報告します。

rstatd カーネルから取得したパフォーマンス統計情報を戻します。

rup ローカル・ネットワーク上のリモート・ホストの状況を示します。

rusers リモート・マシンにログオンしたユーザーのリストを報告します。

rusersd
rusers コマンドからの照会に応答します。

rwall ネットワーク上のすべてのユーザーにメッセージを送信します。

rwalld
rwall コマンドからの要求を処理します。

showmount
リモート・ファイルシステムをマウントしたすべてのクライアントのリストを表示します。

spray 指定された数のパケットをホストへ送信します。

sprayd
spray コマンドが送信したパケットを受け取ります。

以下に示すのは、セキュア・ネットワーキング・デーモンおよびユーティリティーのリストです。

chkey ユーザーの暗号鍵を変更します。

keyenvoy
ユーザー・プロセスとキー・サーバー間の仲介手段を提供します。

keylogin
ユーザーの秘密鍵を暗号化解除および保管します。

keyserv
公開鍵および秘密鍵を保管します。

mkkeyserv
keyserv デーモンを始動し、`/etc/rc.nfs` ファイル内の該当のエントリーをコメントではないようにします。

newkey
公開鍵ファイル内に新規のキーを作成します。

rmkeyserv
keyserv デーモンを停止し、`/etc/rc.nfs` ファイル内の **keyserv** デーモンのエントリーをコメントにします。

ypupdated
ネットワーク情報サービス (NIS) マップ内の情報を更新します。

以下に示すのは、ディスクレス・クライアント・サポート構成ファイルです。

bootparamd
ディスクレス・クライアントへのブートに必要な情報を提供します。

以下に示すのは、NFS サブルーチンのリストです。

cbc_crypt()、**des_setparity()**、または **ecb_crypt()**
Data Encryption Standard (DES) ルーチンをインプリメントします。

LPAR パフォーマンス

このトピックでは、POWER4 ベースのシステムで実行されているパーティションでの AIX のパフォーマンスを検討し、モニターし、チューニングするためのヒントとガイドラインを提供します。

パーティションとそのインプリメンテーションについての詳細は、*AIX 5L™ バージョン 5.3 パーティション環境での AIX インストールまたは ハードウェア管理コンソール インストールおよび操作ガイド*を参照してください。

ロジカル・パーティションに関するパフォーマンスの考慮事項

POWER4 ベースのシステムは、さまざまな方法で構成できます。例えば、大規模なシステムでは Multi Chip Modules (MCM) としてパッケージされた POWER4 CPU を使用し、小規模のシステムでは Single Chip Modules (SCM) としてパッケージされた POWER4 CPU を使用して構成できます。

アプリケーションのワークロードでは、これらのシステムのパフォーマンス上の特性が異なります。

LPAR では、アプリケーション・ソフトウェアが多数のプロセッサー間で十分に拡張できないときや、パーティションの柔軟性が求められるときに、ハードウェアを柔軟に使用できます。 このような場合、複数の小さいパーティションで 1 つのアプリケーションの複数のインスタンスを実行すると、同じアプリケーションの 1 つの大きなインスタンスを実行するときより、優れたスループットが実現できます。 例えば、1 つのアプリケーションが、スレッド化を少ししか、またはまったく行わない 1 つのプロセスとして設計された場合、2-way または 4-way システムでは問題なく実行されますが、大規模な SMP システムで実行する場合には制限があることに気が付きます。 アプリケーションの設計をやり直して、利用できる CPU の数を増やす代わりに、並列の小さい CPU パーティション・セットでアプリケーションを実行できます。

ロジカル・パーティションのパフォーマンスへの影響については、詳細かつ細かい差異分析を行う際に検討する必要があります。 ハイパーバイザーおよびファームウェアは、パーティション用にメモリー、CPU、およびアダプターのマッピングを処理します。 一般に、アプリケーションはパーティションのメモリーがある場所、どの CPU が割り当てられたか、またはどのアダプターが使用中かを認識しません。 アプリケーションのパフォーマンスのモニターとチューニングに関する考慮事項があります。これらの考慮事項は、CPU に対するメモリーの場所、共用 L2 および L3 キャッシュ、およびシステムのパーティション環境を管理するハイパーバイザーのオーバーヘッドに関するものです。

LPAR オペレーティング・システムの考慮事項

LPAR オペレーティング・システムには、幾つかの考慮すべき問題があります。

POWER4 ベース・システム上のパーティションは、次のオペレーティング・システムで実行できます。

- AIX オペレーティング・システム (32 ビット・カーネル)。

- AIX (64 ビット・カーネル)。 AIX 64 ビット・カーネルは、64 ビット・アプリケーションの実行用に最適化され、そのパーティションに割り当てられたサイズの大きな物理メモリーをアプリケーションが使用できるようにすることで、スケーラビリティーを改善します。
- Linux (64 ビット・カーネル)。

システム上の各パーティションはそれぞれ異なるレベルのオペレーティング・システムを実行できます。パーティションは、あるパーティションで実行しているソフトウェアを、他のパーティションで実行しているソフトウェアから分離するように設計されています。 これには、自然に起こるソフトウェア中断や、ソフトウェアが意図的に LPAR バリアを破ろうとする試みに対する保護が含まれます。 通常のネットワーク接続によるアクセス以外の、パーティション間のデータ・アクセスはできません。 1 つのパーティションでソフトウェア・パーティションのクラッシュが起こっても、他のパーティションで、アプリケーション・ソフトウェアとオペレーティング・システム・ソフトウェアの障害などの破損は起こりません。 基礎となるハードウェア共用リソースをパーティションが使用する場合、そのリソースを共用する他のパーティションが使用できるリソースがなくなるほど、多くを使用することはできません。例えば、同じ PCI ブリッジ・チップを共用するパーティションは、バスを永遠にロックすることはできません。

システム・コンポーネント

LPAR 環境をインプリメントし、サポートするには、いくつかのシステム・コンポーネントが協働する必要があります。

プロセッサー、ファームウェア、およびオペレーティング・システム間の関係では、特定の機能がこれらのコンポーネントのそれぞれにサポートされることが必要です。したがって、LPAR のインプリメンテーションは、ソフトウェア、ハードウェア、またはファームウェアのいずれか 1 つに基づくのではなく、3 つのコンポーネント間の関係に基づきます。 POWER4 マイクロプロセッサーは、ハイパーバイザー・モードという拡張形式のシステム・コールをサポートし、これによって、特定のハードウェア機能への特権プログラム・アクセスが可能になります。 このサポートには、プロセッサー内でのこれらの機能の保護も含まれます。 このモードでは、プロセッサーのパーティションのバウンダリーの外側にあるシステムに関する情報に、そのプロセッサーがアクセスできます。 ハイパーバイザーが使用するシステム CPU とメモリー・リソースのパーセンテージはわずかなため、ハイパーバイザーを使用して実行するワークロードを、ハイパーバイザーを使用せずに実行するワークロードと比較しても、わずかな影響しか現れません。

POWER4 ベースのシステムは、以下を含め、さまざまなパーティション構成でブートできます。

- LPAR サポートが実行されていないために、ハイパーバイザーが実行していない、専用ハードウェア・システム。 これは、フル・システム・パーティションと呼ばれます。
- ハイパーバイザーが実行しているシステムで実行するパーティション。

親和性ロジカル・パーティション

一部の POWER プロセッサー・ベースのプラットフォーム システムには、親和性ロジカル・パーティションを作成する機能があります。 この機能は、各パーティションにどのシステム CPU とメモリー・リソースを使用するかを、相互の相対的な物理位置に基づき、自動的に決定します。

ハードウェア管理コンソール (HMC) は、セットアップ処理での管理者の選択に応じて、システムを 4 プロセッサー・パーティションまたは 8 プロセッサー・パーティションを持つ、対称的な LPAR に分割します。 プロセッサーおよびメモリーは、MCM バウンダリー上で位置合わせされます。 この機能は、システムを 1 組の同一クラスター・ノードとして使用できるように設計され、科学的および技術的なワークロードに対してパフォーマンスの最適化を実現します。 このモードでシステムをブートすると、CPU とメモリーを追加および削除することによってリソースをチューニングする機能が使用できません。親和性ロジカル・パーティションで実行するワークロードは、通常のロジカル・パーティションで実行するワークロードよりパフォーマンスの点で優れています。

注: AIX メモリー親和性は、 LPAR モードでは使用できません。

パーティションにおけるワークロード・マネージメント

各 AIX パーティションにも AIX と同じワークロード・マネージメント機能があります。

パーティション内で実行されている AIX ワークロード・マネージャー (WLM) に違いはありません。 WLM は、パーティション間でのワークロードを管理しません。 アプリケーションのオーナーは、CPU またはメモリーをワークロードに指定し、この概念をパーティションにも拡張したい場合があります。 しかしパーティション内では、CPU はワークロード・マネージャーの有効範囲外の各パーティションに割り当てられるため、特定の MCM から特定のワークロードに CPU を指定する機能は使用できません。 この場合でも、ワークロード・マネージャー と **bindprocessor** コマンドは、以前に割り当てられた CPU を特定のワークロードにバインドできます。

パーティションまたはワークロード・マネージメントの選択

特定のワークロード、アプリケーション、またはソリューションに、パーティションを使用するかワークロード・マネージメントを使用するかを選択する際は、いくつか考慮すべき状態があります。

一般に、パーティションは現状が次のような場合に適した管理モードと見なされています。

- 異なるバージョンまたは修正レベルのオペレーティング・システムを必要とする、アプリケーションの依存関係がある場合。
- 異なるオーナー/管理者、機密データの厳格な分離、または間にネットワーク・ファイアウォールがインストールされた分散アプリケーションを必要とするセキュリティー要件がある場合。
- さまざまなリカバリー手順、例えば、**HA** クラスター化とアプリケーション・フェイルオーバー、または異なる災害時回復手順がある場合。
- 障害を厳格に分離し、アプリケーションまたはオペレーティング・システムの障害が相互に影響し合わないようにする必要がある場合。
- パフォーマンスを分離して、ワークロードのパフォーマンス特性が共有リソースに干渉し合わないようにする必要がある場合。

パフォーマンスを分離することは、パーティションをサポートするシステムにおいて、アプリケーションのワークロードをモニターまたはチューニングするときに重要です。 他の重要なワークロードと同時に作業するときは、有効な AIX ワークロード・マネージメント制御を確立するのが困難な場合があります。 複数のアプリケーションをモニターおよびチューニングする方が、細かいリソースをパーティションに割り当てることができる個別のパーティションでは、より実用的です。

LPAR によるパフォーマンスへの影響

LPAR で実行した場合の影響は、同様のプロセッサーを SMP モードで実行した場合と大きな違いはありません。

一般に、システム上で LPAR モードで実行するハイパーバイザー機能が通常のメモリーおよび入出力操作に追加するオーバーヘッドは、5% に達しません。通常、複数のパーティションを同時に実行しても、他のパーティションへのパフォーマンスの影響はほとんどありませんが、パフォーマンスに影響を与える状況があります。 仮想メモリー管理の場合、ハイパーバイザーに関連するオーバーヘッドが多少余分にかかります。 このことは、ほとんどのワークロードではあまり重要ではありませんが、大量のページ・マップ・アクティビティーがある場合にはこの影響が大きくなります。 大規模な SMP システムで十分に拡張されないアプリケーションの場合、別個のパーティションで実行されるワークロード間を厳格に分離することによって、一部のケースではパーティションによって実際にパフォーマンスが改善されます。

小規模システムのシミュレート

パーティションの使用可能メモリーを減らすことにより、より少ない量のメモリーを効果的にシミュレートできます。

rmss コマンドは、POWER4 ベースの MCM システムで使用すると、そのメモリーの MCM に対する位置に関係なく、システムからメモリーを割り当てます。 特定のパフォーマンス特性の詳細は、どのメモリーが使用可能か、およびパーティションにどのメモリーが割り当てられたかに応じて、変化する場合があります。 例えば、**rmss** コマンドを使用して、ローカル・メモリーを使用する 8-way パーティションをシミュレートした場合、実際に割り当てられるメモリーが、MCM に最も近い物理メモリーになる可能性はほとんどありません。 事実、8 つのプロセッサーが MCM 上の 8 つのプロセッサーになる可能性は低く、代わりに、使用可能リストから割り当てられます。

MCM ベースのシステムで CPU の構成を解除すると、MCM とメモリー間のパスを暗黙的に使用するハイパーバイザーには、微妙な影響があります。 パフォーマンスの影響は小さいのですが、詳細なパフォーマンス分析に影響する可能性のある、わずかな変化が生じることがあります。

パーティションにおけるマイクロプロセッサー

マイクロプロセッサーは LPAR に割り当てることができます。

割り当てられたマイクロプロセッサー

LPAR に割り当てられたマイクロプロセッサーのリストを確認するには、HMC の管理対象システム (CEC) を選択し、その属性を表示します。

実行しているパーティションに割り当てられた、すべてのプロセッサーの現行割り当て状態を表示するタブがあります。 AIX はファームウェアが提供した番号を使用します。この番号は、マイクロプロセッサー番号と AIX ロケーション・コードを見て、パーティション内からどのプロセッサーが使用されるかを確認できます。

2 プロセッサー・パーティションに割り当てられたマイクロプロセッサーの状態についての検査は、次のように指定します。

```
 > lsdev -C | grep proc
proc17   Available 00-17   Processor
proc23   Available 00-23   Processor
```

マイクロプロセッサーを使用不可にした場合の影響

MCM を含む POWER4 ベースのシステムでマイクロプロセッサーを使用不可にしても、システム全体にある既存のマイクロプロセッサーを通じて、制御フローとメモリー・アクセシビリティーの経路指定があります。 これはワークロード全体のパフォーマンスに影響します。

パーティション内での仮想プロセッサー管理

カーネル・スケジューラーは、(パーティションの物理的使用率によって測定された) パーティションの即時ロードとともに、仮想プロセッサーの使用を動的に増減するように拡張されました。

各秒単位で、パーティションの物理的使用率に対応して活動化する必要のある仮想プロセッサー数を、このカーネル・スケジューラーが評価します。この数値が高い仮想プロセッサー使用率になっている場合、必要な仮想プロセッサーの基本数を増加して、ワークロードを広範囲の仮想プロセッサーに分散できるようにします。 **schedo** コマンドを使用して、追加の仮想プロセッサーを要求できます。次に、この値を使用して、仮想プロセッサーを使用可能または使用不可にする必要があるのかどうかを判別します。これが必要な理由は、スケジューラーが調整するのは、単に、使用中の仮想プロセッサー数を毎秒 1 つずつ調整するだ

けであるためです。 それで、計算で得られた数が現在活動化されている仮想プロセッサー数より大きい場合、1 つの仮想プロセッサーが活動化されます。 この数が現在活動化されている仮想プロセッサー数より小さい場合は、1 つの仮想プロセッサーが非活動化されます。

仮想プロセッサーが非活動化されると、DLPAR の場合と同様、その仮想プロセッサーはパーティションから動的に除去されます。この除去された仮想プロセッサーは、もう、アンバインドされた作業の実行を続行する対象、またはそれを受け取る対象ではなくなりますが、バインドされたジョブはまだ実行できます。ユーザーまたはアプリケーションが認識できるオンラインの論理プロセッサー数とオンライン仮想プロセッサー数は変わりません。 システム上で稼働するミドルウェアまたはアプリケーションへの影響はありません。アクティブおよび非アクティブの仮想プロセッサーは、システムに対して内部的なものだからです。

vpm_xvcpus チューナブル・パラメーターのデフォルト値は 0 で、これは、フォールディングが使用可能であることを表します。この意味は、仮想プロセッサーは管理対象になっているということです。*vpm_xvcpus* チューナブル・パラメーターを変更するには、schedo コマンドを使用します。

仮想プロセッサー管理機能が使用可能であるかどうかを判別するには、以下のコマンドを使用します。

```
# schedo -o vpm_xvcpus
```

使用中の仮想プロセッサー数を 1 ずつ増やすには、以下のコマンドを使用します。

```
# schedo -o vpm_xvcpus=1
```

各仮想プロセッサーは、最大で 1 つの物理プロセッサーを使用できます。必要な仮想プロセッサー数は、以下の式に示されているとおり、物理的な CPU 使用率と *vpm_xvcpus* チューニング・オプションの値との合計を計算して判別されます。

```
必要な仮想プロセッサー数 =
物理的な CPU 利用状況 + 使用可能にする追加の仮想プロセッサー数
```

必要な仮想プロセッサー数が、使用可能になっている現行仮想プロセッサー数より小さい場合、1 つの仮想プロセッサーが使用不可にされます。 必要な仮想プロセッサー数が、使用可能になっている現行仮想プロセッサー数より大きい場合は、1 つの使用不可の仮想プロセッサーが使用可能にされます。 使用不可の仮想プロセッサーに接続されているスレッドは、依然としてそのプロセッサー上で稼働が可能です。

注: 上記の式から計算された値は、必ず、次に近い整数に切り上げる必要があります。

以下の例は、使用する仮想プロセッサー数の計算方法を説明しています。

最後のインターバルを通して、パーティション A は、2.5 台のプロセッサー数を使用しています。*vpm_xvcpus* チューニング・オプションは 1 に設定されています。 上記の式を使用すると、以下のようになります。

```
物理的な CPU 使用状況 = 2.5
使用可能にする追加の仮想プロセッサー数 (vpm_xvcpus) = 1

必要な仮想プロセッサー数 = 2.5 + 1 = 3.5
```

計算で得られた値を次の整数に切り上げると、4 になります。したがって、システム上で必要な仮想プロセッサー数は 4 です。それで、パーティション A が 8 個の仮想プロセッサーで稼働していた場合、4 つの仮想プロセッサーが使用不可になり、4 個の仮想プロセッサーが使用可能なままとなります。 SMT が使用可能な場合、各仮想プロセッサーは 2 つの論理プロセッサーを獲得します。 したがって、8 つの論理プロセッサーが使用不可になり、8 つの論理プロセッサーが使用可能になります。

以下の例では、フォールディング機能が使用可能になっていない状態で実行中の適度なワークロードは、そのパーティションに割り当てられた各仮想プロセッサーの最小量を消費します。 以下の、4 つの仮想 CPU のあるシステム上での **mpstat -s** ツールからの出力は、仮想プロセッサーの使用率と、それと関連した 2 つの論理プロセッサーの使用率を示します。

```
       Proc0                Proc2                Proc4                Proc6
       19.15%               18.94%               18.87%               19.09%
    cpu0      cpu1       cpu2      cpu3       cpu4      cpu5       cpu6      cpu7
   11.09%    8.07%      10.97%    7.98%      10.93%    7.93%      11.08%    8.00%
```

フォールディング機能が使用可能になっている場合、システムは、上記の式を使用して、必要な仮想プロセッサー数を計算します。 計算で得られた値は、パフォーマンス低下なしに、適度なワークロードを実行するのに必要な仮想プロセッサー数を減らすのに使用されます。 以下の、4 つの仮想 CPU のあるシステム上での **mpstat -s** ツールからの出力は、仮想プロセッサーの使用率と、それと関連した 2 つの論理プロセッサーの使用率を示します。

```
       Proc0                Proc2                Proc4                Proc6
       54.63%               0.01%                0.00%                0.08%
    cpu0      cpu1       cpu2      cpu3       cpu4      cpu5       cpu6      cpu7
   38.89%   15.75%      0.00%     0.00%       0.00%     0.00%      0.03%     0.05%
```

上記のデータから分かるように、このワークロードが恩恵を受けるのは、この作業が 1 つの仮想プロセッサーに集中時に、使用率の低減と補助となるプロセッサーの保守の削減、および親和性の増大の点です。ただし、このワークロードが大きい場合は、フォールディング機能はすべての仮想 CPU を使用する能力に介入しません (必要な場合も)。

アプリケーションに関する考慮事項

LPAR に関連したアプリケーションに関して、幾つかの注意すべき項目があります。

一般に、アプリケーションはそれ自体が LPAR で実行していることを認識しません。管理者が認識できるわずかな違いがありますが、これらは、アプリケーションからマスクされています。 これらの考慮事項とは別に、AIX はスタンドアロン・サーバーで実行する場合と同じ方法で、パーティション内でも実行します。 アプリケーションまたは管理者から見て、違いは見受けられません。 LPAR は AIX アプリケーションおよびたいていの AIX パフォーマンス・ツールに対して透過的です。サード・パーティー・アプリケーションは、AIX のレベルについてだけ認証する必要があります。

LPARでの uname コマンドの実行

LPAR に関連して、システムに関する情報を取得する場合は、**uname** コマンドを使用します。

```
> uname -L
-1 NULL
```

「-1」は、システムはロジカル・パーティションを使用して実行されていないが、フル・システム・パーティション・モードで実行されていることを示します。

以下の例では、**uname** コマンドがどのように、 HMC が管理するパーティション番号とパーティション名を戻すかを示します。

```
 > uname -L
3 Web Server
```

アプリケーションが LPAR モードで実行していることを知っていると、わずかなパフォーマンスの違いを評価するときに役立ちます。

仮想コンソール

各パーティションには物理コンソールはありません。

物理シリアル・ポートをパーティションに割り当てることはできますが、一度に 1 つのパーティションに割り当てられるのは 1 つだけです。 診断の目的およびコンソール・メッセージの出力を提供するために、ファームウェアは仮想 tty をインプリメントします。これは、AIX では標準の tty デバイスとして判断されます。 仮想 tty 出力は HMC にストリームされます。 AIX 診断サブシステムは仮想 tty をシステム・コンソールとして使用します。 パフォーマンス上の観点では、システム・コンソールに多くのデータが書き出され、それが HMC のコンソールでモニターされている場合、HMC への接続は、シリアル・ケーブル接続に限定されます。

時刻 (TOD) クロック

各パーティションにはそれぞれ独自の時刻機構 (TOD) 値があり、パーティションが別々のタイムゾーンで動作できるようになっています。

パーティション同士が通信するための唯一の手段は、標準のネットワーク接続による方法です。 システム上の各パーティションからのトレース情報またはタイム・スタンプ付き情報を参照すると、パーティションの構成方法によって各タイム・スタンプは異なります。

システムのシリアル番号

uname -m コマンドは、パーティションが定義されたときのさまざまなシステム情報を提供します。

シリアル番号はシステムのシリアル番号のことで、すべてのパーティションで同じです。 各パーティションでは、同じシステム・シリアル番号が表示されます。

メモリーに関する考慮事項

メモリーを処理するときに考慮すべき問題がいくつかあります。

パーティションは、「必須」、「望ましい」、および「最小限度」のメモリーを指定して定義されます。システム・リブート後に変化したパフォーマンス条件を評価するときは、メモリーと CPU の割り当てが、基礎となるリソースの可用性に基づいて変更された可能性があることを知ることが重要です。 さらに、HMC からパーティションに割り当てられたメモリー量が、割り当てられた合計であることを覚えておいてください。 パーティション内では、その物理メモリーの一部が、ハイパーバイザー・ページ・テーブル変換サポートで使用されます。

メモリーは、システム全体でシステムによって割り当てられます。 パーティション内のアプリケーションは、メモリーが物理的に割り当てられた場所を判断できません。

動的ロジカル・パーティショニング

DLPAR は、2002 年 10 月以降のマイクロコード更新が適用された POWER4 ベースの System p システムで使用可能です。 いろいろなパーティションをさまざまなレベルのオペレーティング・システムで実行することができます。

DLPAR を使用可能にするに前に、追加のリソースをシステムに追加するためにパーティションをリブートしました。 DLPAR は、アクティブなロジカル・パーティションからプロセッサー、メモリー、入出力スロット、入出力ドロワーを動的に追加および除去できるようにすることによって、論理的にパーティションに分割されたシステムの柔軟性を高めます。 パーティションの可用性に影響を与えることなく、ハードウェア・リソースの再割り当てや、システム容量への要求の変化に対応する調整を行うことができます。

DLPAR を使用して、以下に示す基本的な操作を実行できます。

- リソースをあるパーティションから他のパーティションに移動します。
- パーティションからリソースを除去します。
- パーティションにリソースを追加します。

パーティションに割り当てられていないプロセッサー、メモリー、入出力スロットは、「フリー・プール」に存在します。 システム上の既存のパーティションは、そのシステム上の他のパーティションやフリー・プールから見えません。 DLPAR を使用してプロセッサーをアクティブ・パーティションから除去すると、システムはそのプロセッサーを解放してプールに入れます。その状態であれば、このプロセッサーをアクティブ・パーティションに追加できます。 プロセッサーがアクティブ・パーティションに追加された時点では、パーティションのメモリー、入出力アドレス・スペース、および入出力割り込みのすべてにアクセスが可能です。 このプロセッサーは、パーティションのワークロードに完全に組み込まれます。

256 MB のメモリー領域またはチャンクでは、メモリーの追加または除去を実行できます。 AIX カーネルはほとんど仮想モードで実行しているので、 AIX パーティション内で実行しているアプリケーションに対するメモリー除去の影響は最小限ですみます。 アプリケーション、カーネル・エクステンション、およびほとんどのカーネルは、仮想メモリーのみを使用します。 メモリーを除去すると、パーティションはページングを開始する可能性があります。 AIX カーネルの一部はページング可能であり、これによってパフォーマンスが悪化する可能性があります。 メモリーを除去する際には、ページングが発生していないことを確認するために、ページング統計情報をモニターしてください。

ネットワーク・アダプター、CD ROM デバイス、テープ・ドライブなどの入出力スロットをアクティブ・パーティションで追加および除去できます。 したがって、複数のパーティションに対応するために、頻繁には使用しない物理装置を幾つも購入してインストールしなければならないという問題がなくなります。 プロセッサーやメモリーの追加または除去とは異なり、入出力スロットを再構成するには、アクティブ・パーティションでデバイスの追加や除去を行う前に、特定の PCI ホット・プラグ手順を実行する必要があります。 ホット・プラグ手順は、SMIT によって実行できます。

ハードウェア管理コンソール (つまり HMC) は、システムに接続されており、これにより、動的再構成 (DR) オペレーションの実行が可能になります。 HMC は、 DLPAR をサポートするために R3V1.0 を実行する必要があります。 DLPAR に関連する HMC オペレーションのリストについては、「*IBM eServer pSeries Servers* のための完全パーティション・ガイド」を参照してください。

ハイパーバイザーは、ハードウェア管理機能と、単一物理システム上で動作する仮想マシン (パーティション) の分離機能を提供する薄いソフトウェア層です。 パーティション間のリソース移動を制御するためのコマンドは、 HMC グラフィック・ユーザー・インターフェースまたは HMC コマンド・ラインから LPAR ハイパーバイザーに渡すことができます。 ハイパーバイザーの 1 つのインスタンスのみを実行でき、そのハイパーバイザーだけが、システム・リソースの確認と割り当てを行う機能を持ちます。 DLPAR は、パーティションのセキュリティーを損ないません。 パーティション間で移動されるリソースは再初期化されるので、残余データは発生しません。

DLPAR のパフォーマンスへの影響

DLPAR パフォーマンスの改善または低下により、さまざまな影響があります。

複数の論理メモリー・ブロックでメモリーの追加または除去を行えます。 パーティションからメモリーを除去する場合、 DLPAR オペレーションの完了に要する時間は、除去するメモリー・チャンクの数に関連しています。 例えば、4GB のメモリーをアイドル・パーティションから取り外す DR オペレーションは、1 分から 2 分ですみます。しかし、動的パーティションの大規模メモリー・ページは、サポートされていません。 ラージ・ページを含むメモリー領域は除去できません。

親和性ロジカル・パーティション構成は、マルチ・チップ・モジュール (MCM) バウンダリーに基づく固定パターンに CPU およびメモリー・リソースを割り当てます。 HMC は親和性パーティション上で DR プロセッサーやメモリーのサポートを提供しません。 親和性ロジカル・パーティションの実行時には、入出力アダプター・リソースの動的再構成のみが可能です。

アプリケーションやミドルウェアを DLPAR 対応にすることによっても、動的リソース割り振りおよび割り振り解除の利点を活用できます。これはつまり、アプリケーションが新しいハードウェア・リソースに対応して、自身のサイズ変更を行うということです。 AIX には、ベンダー・アプリケーションを動的にサイズ変更するための DLPAR スクリプトと API が用意されています。 これらのスクリプトまたは API の使用法については、プログラミングの一般概念の『DLPAR』のセクションを参照してください。

DLPAR チューニング・ツール

DLPAR をモニターおよびサポートする場合に使用できる幾つかのツールがあります。

DLPAR を使用して、オンライン・プロセッサーの数を動的に変更できます。 システムのオンライン・プロセッサーの数と最大可能プロセッサーの数の違いを追跡するために、以下のパラメーターを使用できます。

_system_configuration.ncpus
オンライン・プロセッサーの数を照会します。

_system_configuration.max_ncpus
システムの最大可能プロセッサー数を提供します。

プロセッサーおよびメモリーの追加および除去をキャプチャーするためのトレース機能を有効にするために、AIX は、トレース・フック **38F** をサポートしています。

curt、**splat**、**filemon**、**netpmon**、**tprof**、**pprof** などのパフォーマンス・モニター・ツールは、DR アクティビティーを処理するように設計されていません。 これらは、静的プロセッサーまたはメモリーの量に依存しています。 いくつかのケースでは、DR オペレーションはツールの実行による悪影響を受けません。 **tprof** や **pprof** ツールがこれにあたります。しかし、例えば **curt** ツールなどを実行した場合には、DR オペレーションは未定義の動作をしたり、誤った結果を引き起こしたりします。

DR オペレーションをサポートするツールがあります。 これらのツールは、構成変更を検出し、それに応じてレポートを調整するように設計されています。 DLPAR サポートを提供するツールは、以下のとおりです。

topas インターフェースおよび出力の解釈に変更はありません。 **topas** ツールは、DR オペレーションの実行時に、リソースの追加または除去を検出し、新しいリソースのセットに基づいた適切な統計情報をレポートします。

sar、**vmstat**、**iostat**
これらのコマンドのインターフェースおよび出力の解釈に変更はありません。 DR オペレーションを実行すると、構成変更を知らせるメッセージが送られてきます。 ツールはその後、新しいリソースのセットに基づいた適切な統計情報のレポートを開始します。

rmss このコマンドの起動方法に変更はありません。 **rmss** ツールの動作中に DR オペレーションが発生した場合でも、継続して期待どおりに動作します。

マイクロプロセッサーまたはメモリーの追加に関する DLPAR のガイドライン

メモリーまたはプロセッサーの不足が発生する場合、または入出力スロットを追加する必要がある場合は、幾つかの判別方法があります。

パーティションからメモリーを除去する場合、除去するメモリーを吸収するだけの十分な空き物理メモリーが存在しなくても、物理メモリーの代わりになる十分なページング・スペースが存在すれば、DR オペレーションは成功します。しかし、DR メモリーの除去の前と後にパーティションのページング統計情報をモニターすることは重要です。 ページングを操作するために仮想メモリー・マネージャーが用意されていますが、過大なページングはパフォーマンスの低下につながる可能性があります。

メモリーまたはプロセッサーの不足の発生時期を判断するために、 138 ページの『メモリー・パフォーマンス』および 112 ページの『マイクロプロセッサーのパフォーマンス』のガイドラインを参照してください。 入出力スロットを追加しなければならない時期を判断するために、 279 ページの『ネットワーク・パフォーマンス』のガイドラインを参照してください。 これらのガイドラインを使用して、いずれかのリソースを削減した場合の影響を推定することも可能です。

Micro-Partitioning

ロジカル・パーティションにより、同一システムで複数のオペレーティング・システムを干渉し合わずに実行できます。AIX の旧バージョンでは、異なるパーティション間でプロセッサーを共用することはできませんでした。マイクロ・パーティショニングとも呼ばれる共用プロセッサー・パーティション、すなわち SPLPAR を使用することができます。

マイクロ・パーティショニング について

マイクロ・パーティショニングでは、仮想プロセッサーを物理プロセッサーにマップし、その仮想プロセッサーは、物理プロセッサーではなく、パーティションに割り当てられます。

Hypervisor を使用して、共用パーティションに対して認可するプロセッサー使用量のパーセンテージを指定できます。これは、ライセンスとして定義されます。最小のプロセッサー・ライセンスは 10% です。

マイクロ・パーティショニングを使用すると、以下の利点があります。

- 最適なリソース使用率
- 新規サーバーの迅速な配備
- アプリケーションの分離

マイクロ・パーティショニング は、System p5® システムで使用できます。 いろいろなパーティションをさまざまなレベルのオペレーティング・システムで実行することが可能ですが、マイクロ・パーティショニング は、AIX バージョン 6.1 以降を実行するパーティション上でのみ使用できます。

IBM eServer p5 サーバーでは、以下のタイプのパーティションを ハードウェア管理コンソール、つまり HMC から選択できます。

専用プロセッサー・パーティション

専用プロセッサー・パーティションを使用した場合、プロセッサー全体が特定のロジカル・パーティションに割り当てられます。

また、パーティション上の処理容量は、そのパーティション内に構成されているプロセッサーの合計処理容量で制限されるため、この容量を超えることはできません。ただし、例外としては DLPAR 操作を使用してパーティション内にプロセッサーをさらに追加する場合は、その限りではありません。

共用プロセッサー・パーティション

共用プロセッサーを使用した場合、物理プロセッサーは仮想化されてから、パーティションに割り当てられます。

仮想プロセッサーは、物理プロセッサーの 10% から最大でプロセッサー全体までの範囲の能力容量を保有します。 したがって、1 つのシステムでは複数のパーティションを保有でき、その各パーティションが同じプロセッサーを共用し、各パーティション自身の間で処理能力を分割します。 パーティションあたりの仮想プロセッサーの最大数は 64 です。 詳しくは、 392 ページの『パーティション内での仮想プロセッサー管理』を参照してください。

マイクロ・パーティショニング のインプリメンテーション

LPAR と同様に、HMC を使用して マイクロ・パーティショニング にパーティションを定義することができます。

以下の表は、マイクロ・パーティショニング で使用できるさまざまなプロセッサー・タイプをリストしたものです。

プロセッサーのタイプ	説明
物理プロセッサー	物理プロセッサーとは、固有のプロセッサー・コアの数を表す実ハードウェア・リソースであり、プロセッサー・チップ数ではありません。 各チップには、2 つのプロセッサー・コアが含まれています。物理プロセッサーの最大数は、POWER5 ベースのシステム上では 64 です。
論理プロセッサー	論理プロセッサーとは、オペレーティング・システムの視点から見た管理対象の処理装置です。 論理プロセッサーの最大数は 128 です。
仮想プロセッサー	仮想プロセッサーとは、異なるパーティションが共用する論理プロセッサーのパーセンテージを示す装置です。 仮想プロセッサーの最大数は 64 です。

パーティション作成時に共用プロセッサー・パーティションまたは専用プロセッサー・パーティションのどちらを作成するかを選択する必要があります。 1 つのパーティション内に共用プロセッサーと専用プロセッサーの両方を持つことはできません。 プロセッサーの共用を使用可能にするには、以下のオプションを構成する必要があります。

- プロセッサー共用モード: 上限あり (Capped) または上限なし (Uncapped) [1]
- 処理能力: 重み付け [2]
- 仮想プロセッサー数: 必要、最小、および最大

注: 上限ありモードは、処理能力が、割り当てられた能力を決して超えないことを意味し、上限なしモードは、共用処理プールに使用可能なリソースがあるときは処理能力を超えられることを意味します。

注: この処理能力は処理単位という表現で指定され、この単位は 1 つのプロセッサーを 0.01 で細分化した単位で測定されます。 したがって、例えば、プロセッサーの半分について処理能力を割り当てる場合、HMC 上で 0.50 処理単位を指定する必要があります。

マイクロ・パーティショニング のパフォーマンスへの影響

マイクロ・パーティショニング を使用すると、パフォーマンスに対してプラス・マイナス両方の影響があります。

マイクロ・パーティショニング の利点は、各パーティションが必要とするプロセッサー・リソース必要量のみを適用することにより、システム・リソースの全体的な使用率を改善できることです。 しかし、オンライン状態の仮想プロセッサーを維持することに関わるオーバーヘッドがあるため、この属性値の選択時はこのオーバーヘッドの能力要件を考慮する必要があります。

最適なパフォーマンスを得るために、必ず、最小量のパーティションを作成します。こうすると、仮想プロセッサーをスケジューリングするオーバーヘッドを削減できます。

高性能コンピューティング・アプリケーションなど、CPU 集中アプリケーションは、マイクロ・パーティショニング 環境に適さない場合があります。アプリケーションが実行中にそのライセンス済みの処理能力のほとんどを使用する場合は、専用プロセッサー・パーティションを使用してアプリケーションの要求を処理する必要があります。

Active Memory Expansion (AME)

Active Memory™ Expansion (AME) は、システムの有効なメモリー容量を拡張するための新テクノロジーです。 AME では、メモリー内のデータを透過的に圧縮するためのメモリー圧縮テクノロジーを採用しています。これにより、より多くのデータをメモリーに入れることが可能となり、構成されたシステムのメモリー容量を拡張することができます。

概説

Active Memory Expansion (AME) は、メモリー内データを圧縮することにより、メモリーに入れることができるデータ量を増加させ、それにより IBM Power Systems プロセッサー・ベースのサーバーの有効メモリー容量を拡張します。このメモリー内データ圧縮はオペレーティング・システムが管理するため、アプリケーションおよびユーザーはこの圧縮を意識する必要がありません。 AME は論理区画 (LPAR) 単位に構成できます。 このため、AME はシステム上の 1 つ以上の LPAR を選択して使用可能にできます。

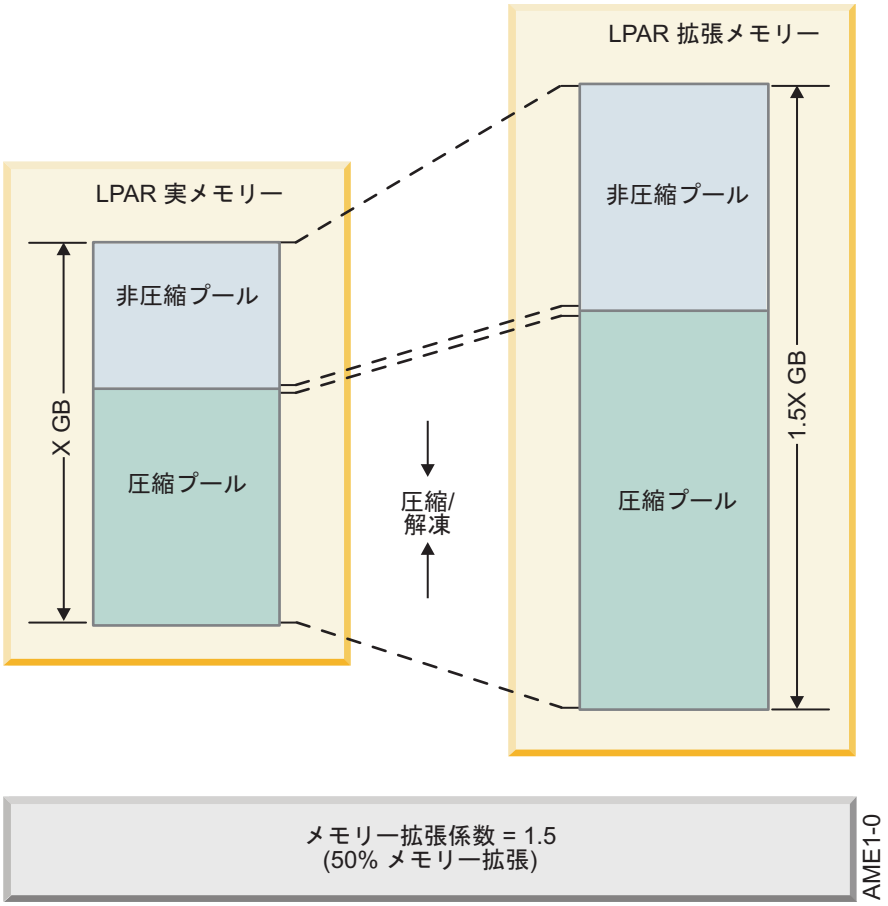
ある LPAR に対して Active Memory Expansion を使用可能にすると、オペレーティング・システムはその LPAR のメモリーの一部を圧縮し、残りの部分は圧縮されていない状態のままにします。この結果、メモリーが事実上 2 つのプールに分けられます。 このプールは以下の 2 つです。

- 圧縮プール
- 非圧縮プール

オペレーティング・システムは、LPAR のワークロードとその構成に基づいて、圧縮するメモリー量を動的に変更します。

オペレーティング・システムは、アプリケーションのメモリー・アクセス・パターンに基づいて、圧縮メモリー・プールと非圧縮メモリー・プール間でデータを移動させます。 アプリケーションが圧縮されたデータにアクセスする必要がある場合、オペレーティング・システムはそのデータを自動的に解凍し、圧縮プールから非圧縮プールに移動させます。それによってアプリケーションはそのデータを使用できるようになります。非圧縮プールがフル状態になると、オペレーティング・システムはデータを圧縮して非圧縮プールから圧縮プールにデータを移動させます。

アプリケーションは、この圧縮と解凍のアクティビティーを意識する必要がありません。 AME はメモリー圧縮に依存して行われますので、 AME が使用状態になると、新たな CPU 利用が若干必要となります。 AME 用に必要となる追加の CPU 使用量は、使用されているワークロードとメモリー拡張のレベルに従って異なります。



注: AME が使用可能になると、AIX オペレーティング・システムは 4 KB ページを使用します。ただし、IBM AIX 7.2 (テクノロジー・レベル 1) 以降が POWER8® システム上で稼働している場合、 **vmo** コマンドを `ame_mpsize_support` パラメーターと一緒に使用すると、64 KB ページ・サイズを有効にできます。

メモリー拡張係数および拡張メモリー・サイズ

Active Memory Expansion の構成時、LPAR に対して設定すべき構成オプションが 1 つだけあり、それがメモリー拡張係数です。 LPAR のメモリー拡張係数には、その LPAR に対するターゲットの有効メモリー容量を指定します。 このターゲット・メモリー容量により、メモリー圧縮で使用可能となるメモリー量をオペレーティング・システムに指示します。 指定されたターゲット・メモリー容量は、拡張メモリー・サイズと呼ばれます。 メモリー拡張係数は、LPAR の実メモリー・サイズの乗数として指定します。

```
LPAR_expanded_mem_size = LPAR_true_mem_size * LPAR_mem_exp_factor
```

例えば、ある LPAR に対してメモリー拡張係数 2.0 を使用すると、その LPAR のメモリー容量を倍にするようにメモリー圧縮は使用される必要があることを示します。 メモリー拡張係数 2.0 とメモリー・サイズ 20 GB を使用して LPAR を構成すると、その LPAR 用の拡張メモリー・サイズは 40 GB となります。

```
40 GB = 20 GB * 2.0
```

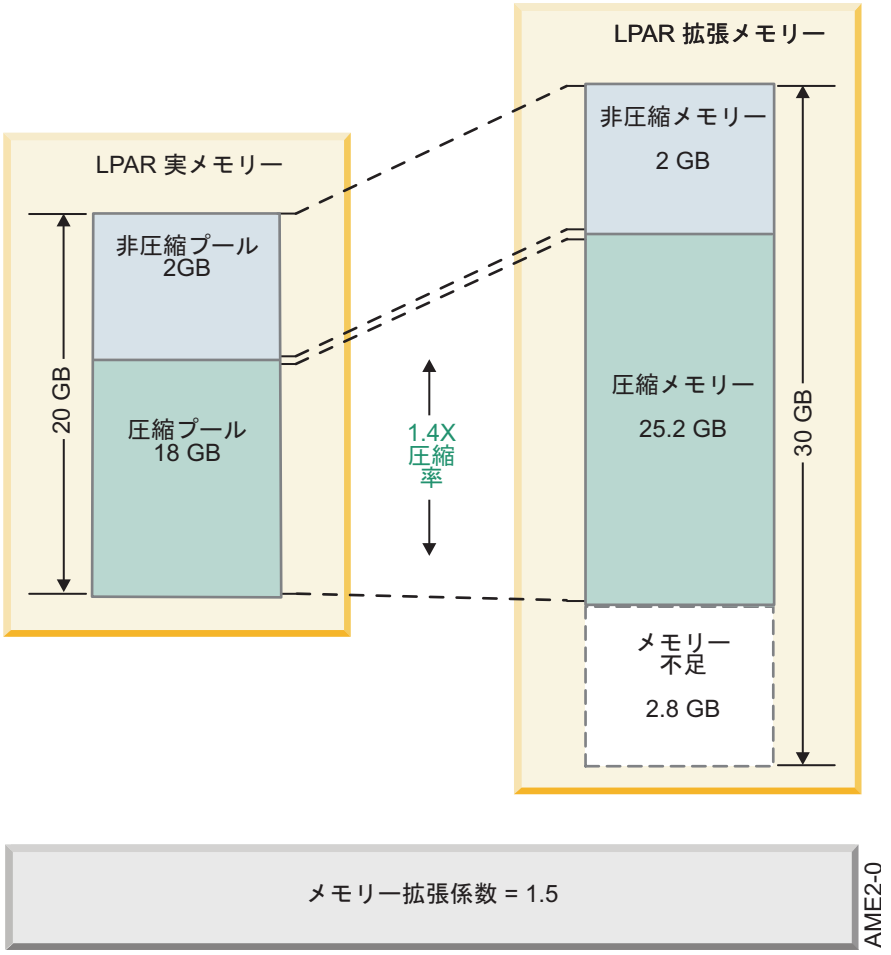
オペレーティング・システムは、 40 GB のデータが 20 GB のメモリーに入るように、メモリー内データを圧縮します。 メモリー拡張係数と拡張メモリーサイズは実行時に動的に変更できます。このためには、ハードウェア管理コンソール (HMC) で動的 LPAR 操作を行います。 拡張メモリー・サイズは、最も近

い論理メモリー・ブロック (LMB) の倍数に、常時切り捨てられます。

メモリー不足

LPAR に対してメモリー拡張係数を構成する場合、ワークロードがどれだけ圧縮できるかにより、メモリー拡張係数を過大に指定してしまい、達成できない場合があります。 LPAR に対してメモリー拡張係数を過大に指定すると、メモリー拡張不足となります。すなわち、その LPAR は、指定されたメモリー拡張係数のターゲットを達成できないことを示します。 例えば、メモリー・サイズ 20 GB、メモリー拡張係数が 1.5 で LPAR を構成していると、合計のターゲット拡張メモリー・サイズは 30 GB となります。 しかし、その LPAR で実行するワークロードが十分に圧縮されず、ワークロードのデータは 1.4 から 1 の間の割合でしか圧縮されないとします。 この場合、ワークロードはターゲットのメモリー拡張係数 1.5 を実現できません。 オペレーティング・システムは、圧縮プールで使用可能な物理メモリー量を最大 95% に制限します。この例で、圧縮プールに 19 GB が確保されているとするならば、実現可能な拡張メモリー・サイズは最大 27.6 GB となります。 この結果、 2.4 GB が不足分となります。 この不足分はメモリー不足と呼ばれます。

このメモリー不足の影響は、非常に少ないメモリーで LPAR を構成した場合の影響と同じです。 メモリー不足が発生すると、オペレーティング・システムは LPAR に対して構成された拡張メモリー・ターゲットを達成できず、仮想メモリー・ページをページング・スペースにページアウトすることに頼る必要が生じる可能性があります。 このようにして、前述した例の中では、ワークロードが 27.6 GB よりも多くのメモリーを使用すると、オペレーティング・システムは仮想メモリーをページング・スペースにページアウトし始めることになります。 ワークロードがその拡張メモリー・サイズを獲得可能かどうかの兆候を知るために、オペレーティング・システムはメモリー不足メトリックをレポートします。 これは、獲得できない拡張メモリー・サイズ内の「欠陥部分」です。 この不足がゼロの場合、ターゲット・メモリー拡張係数が達成されて、LPAR のメモリー拡張係数は正しく構成されていることになります。 拡張メモリーの不足メトリックがゼロ以外の場合、ワークロードは不足サイズ分だけ、拡張メモリー・サイズを獲得できないことになります。 メモリー不足が発生しないようにするには、LPAR のメモリー拡張係数を減少させる必要があります。 ただし、メモリー拡張係数を減少させると、LPAR の拡張メモリー・サイズも減少します。 その結果、LPAR の拡張メモリー・サイズを同じ量に保持するには、メモリー拡張係数を減少させる必要があり、より多くのメモリーを LPAR に追加する必要があります。 LPAR のメモリー・サイズとメモリー拡張係数はいずれも動的に変更可能です。



計画上の考慮点

Active Memory Expansion (AME) 環境でワークロードをデプロイする前に、若干の初期計画を行う必要があり、それによってワークロードが AME から最大の利益が得られるようにします。ワークロードに対して AME が提供する利益は、そのワークロードの特性によって異なります。 あるワークロードは、他のワークロードに比べて高いレベルのメモリー拡張を実現できます。 Active Memory Expansion Planning and Advisory Tool (**amepat**) は、Active Memory Expansion 環境におけるワークロードのデプロイメント計画をワークロードが実現可能なメモリー拡張のレベルに関してガイドとなる情報を提供します。

AME 計画ツール

AME Planning Tool (`/usr/bin/amepat` にある) には、2 つの主な目的があります。 すなわち、

- 初期の Active Memory 構成の計画
- アクティブな AME 構成のモニターと微調整

AME Planning Tool は LPAR 上で稼働し、その時に AME が使用可能であっても使用不能であっても構いません。 AME が使用可能になっていない LPAR では、**amepat** は代表的なワークロードを使って実行させてください。 その場合、重要な時間帯に対応したワークロードをモニターするように設定する必要があります。 例えば、ワークロードがリソースをピーク状態で使用する時間帯に対して、**amepat** ツールを実行するように設定します。 このツールが終了すると、可能性がある複数のメモリー拡張係数のレポート、および AME に起因する予想 CPU 使用率を各メモリー拡張係数ごとにレポート表示します。 また、このツールは、推奨されるメモリー拡張係数を提供します。この係数を使用すると、追加で必要となる

CPU 使用率を最小化しながら、メモリーの節約を最大化するように努めます。このレポートと推奨値は、AME 展開での初期構成時に有効に使用できます。 AME が使用可能状態の LPAR では、**amepat** は同様の目的を果たします。代表的なワークロードに対してピーク時にこのツールを実行すると、現行のメモリー拡張係数での AME に起因する実 CPU 使用率がレポートされます。 また、このツールはメモリー不足情報も表示します (メモリー不足状態があった場合)。 AME が使用可能であるため、種々のメモリー拡張係数で CPU 使用率のレベルがどのようになる可能性があるかを、このツールは一層正確に表示することもできます。 この情報に基づく新しい推奨値が、ユーザーに提供されることになります。

AME が使用不能状態の区画で生成されたレポートのサンプル、およびサンプルのワークロード例は、以下のとおりです。

```
# amepat 5 2

Command Invoked                : amepat 2 5

Date/Time of invocation        : Wed Dec  2 11:29:29 PAKST 2009
Total Monitored time           : 10 mins 58 secs
Total Samples Collected        : 5

System Configuration:
---------------------
Partition Name                 : aixfvt19
Processor Implementation Mode  : POWER5
Number Of Logical CPUs         : 8
Processor Entitled Capacity    : 4.00
Processor Max. Capacity        : 4.00
True Memory                    : 4.25 GB
SMT Threads                    : 2
Shared Processor Mode          : Disabled
Active Memory Sharing          : Disabled
Active Memory Expansion        : Disabled

System Resource Statistics:    Average       Min         Max
---------------------------    -----------   ----------  -----------
CPU Util (Phys. Processors)    2.00 [ 50%]   1.00 [ 25%] 3.00 [ 75%]
Virtual Memory Size (MB)       1366 [ 31%]   1113 [ 26%] 2377 [ 55%]
True Memory In-Use (MB)        1758 [ 40%]   1234 [ 28%] 3834 [ 88%]
Pinned Memory (MB)             673 [ 15%]     673 [ 15%]  675 [ 16%]
File Cache Size (MB)           391 [  9%]     124 [  3%] 1437 [ 33%]
Available Memory (MB)          841 [ 65%]    1812 [ 42%] 3099 [ 71%]

Active Memory Expansion Modeled Statistics
------------------------------------------
Modeled Expanded Memory Size   :   4.25 GB
Average Compression Ratio      :   5.29

Expansion    Modeled True      Modeled              CPU Usage
Factor       Memory Size       Memory Gain          Estimate
---------    -------------     ------------------   -----------
     1.00          4.25 GB         0.00 KB [  0%]   0.00 [  0%]
     1.31          3.25 GB         1.00 GB [ 31%]   0.34 [  8%]
     1.55          2.75 GB         1.50 GB [ 55%]   0.39 [ 10%]
     1.89          2.25 GB         2.00 GB [ 89%]   0.45 [ 11%]
     2.12          2.00 GB         2.25 GB [112%]   0.50 [ 12%]
     2.43          1.75 GB         2.50 GB [143%]   0.65 [ 16%]
     2.83          1.50 GB         2.75 GB [183%]   0.70 [ 18%]

Active Memory Expansion Recommendation:
---------------------------------------
The recommended AME configuration for this workload is to configure the LPAR
with a memory size of 1.50 GB and to configure a memory expansion factor
of 2.83.  This will result in a memory gain of 183%. With this
configuration, the estimated CPU usage due to AME is approximately 0.50
physical processors, and the estimated overall peak CPU resource required for
```

```
the LPAR is 3.50 physical processors.

NOTE: amepat's recommendations are based on the workload's utilization level
during the monitored period. If there is a change in the workload's utilization
level or a change in workload itself, amepat should be run again.

The modeled Active Memory Expansion CPU usage reported by amepat is just an
estimate.  The actual CPU usage used for Active Memory Expansion may be lower
or higher depending on the workload.
```

このレポートは、以下に順次説明される 6 つのセクションで構成されます。

コマンド情報

このセクションでは、**amepat** に渡された引数、起動時刻、システムがモニターされた合計時間、および収集されたサンプル数に関する情報が表示されます。上記のレポートでは、**amepat** は

10 分間起動され、2 分間隔で 5 回のサンプリングを行っています。

注**:** 表示された「Total Monitored time」は、10 分 58 秒となっています。余分な 58 秒は、Active Memory Expansion Modeling に必要となるシステム統計の収集に使用されています。

「System Configuration」(システム構成)

このセクションでは、構成情報 (ホスト名、プロセッサー・アーキテクチャー、CPU 数、ライセンス、実メモリー・サイズ、SMT 状態、プロセッサー、およびメモリー・モードなど) が表示されます。 上記のレポートでは、Active Memory Expansion の使用不可状態は、**amepat** が AME 使用不可区画で起動されたことを示しています。

注**:** また、**amepat** ツールは AME 使用可能区画で起動して、アクティブ AME 構成のモニターと微調整を行うこともできます。 AME 使用可能区画の場合、ターゲット拡張メモリー・サイズとターゲット・メモリー拡張係数も「System Configuration」セクションに表示されます。

「System Resource Statistics」(システム・リソース統計)

このセクションでは、モニター処理期間中のシステム・リソース使用状況が表示されます。ここでは、システム・リソース使用状況として平均、最小、および最大値と、それに対応したパーセントを表示します。

上記のレポートでは、平均としてのワークロードは、2.00 物理プロセッサー (「CPU Util」行参照) を使用しています。この数値は、LPAR の最大物理能力の 50% (「Processor Max Capacity」行に表示された 4.00) にあたります。

注**:** CPU 使用状況レポートには、AME モデリング用に利用された処理単位数が含まれています。 AME が使用可能な区画では、このレポートには圧縮/解凍アクティビティーで使用された CPU 使用状況も含まれます。

上記のレポートでは、ワークロードのメモリー使用状況も表示することができます。 すべてのメモリーのパーセンテージ値は、LPAR の合計実メモリーを 100 とした場合の割合です。

注**:** 「Pinned Memory」と「File Cache」が高いパーセント値の場合、そのワークロードは AME を使用しても大きなメリットがない可能性があることを示しています。

Active Memory Expansion 統計情報

このセクションが表示されるのは、AME が使用可能な LPAR で **amepat** ツールが起動された場合に限定されます。

以下にその出力例を示します。

```
AME Statistics:                  Average        Min       Max
---------------                 -----------    --------   ---------
AME CPU Usage (Phy. Proc Units) 0.25 [  6%]   0.01 [  0%] 0.50 [ 13%]
Compressed Memory (MB)          264 [ 13%]    264 [ 13%]  264 [ 13%]
Compression Ratio               2.15          2.15        2.16
Deficit Memory Size (MB)        562 [ 55%]    562 [ 55%]   562 [ 55%]

This section of the report shows the AME CPU Usage, Compressed Memory,
Compression Ratio & Deficit Memory.
The Deficit Memory will be reported only if there is a memory deficit
in achieving the expanded memory size.
Otherwise the tool will not report this information.
For example in the above report, it can be observed that there is
an average memory deficit of 562 MB which is 55% of the Target
Expanded Memory Size of 2 GB (which is reported in the
System Configuration Section when AME is enabled).
The report also shows on an average 264 MB out of 2GB of expanded memory is in
compressed form as reported against Compressed Memory.
```

Active Memory Expansion モデル化統計情報

このセクションでは、Modeled Expanded Memory サイズ、圧縮率、および可能性のある複数の AME 構成を持ったテーブルが表示されます。 前述のレポートでは、Modeled Expanded Memory サイズは、

4.25 GB としてレポートされています。この数値は LPAR の実メモリー・サイズです。 デフォルトでは、**amepat** は「Modeled Expanded Memory」として区画の実メモリーを使用します。 これは、

-t または **-a** オプションを使用して変更可能です。 このレポートでは平均圧縮率 5.29 を表示していますが、この数値はワークロードがうまく圧縮していることを示しています。 圧縮率が 1 に近いほど、

メモリー拡張がほとんど不可能なことを示しています。 モデリング・テーブルに表示された構成は、ターゲット・メモリーとして「Modeled Expanded Memory Size」に基づきます。

このテーブルには、「Modeled True Memory Size」、「Modeled Memory Gain」、および AME に起因する追加の CPU 使用が、種々の拡張係数ごとに表示されます。

例えば、このテーブルの次の行を見てください。

```
1.55          2.75 GB         1.50 GB [ 55%]   0.39 [ 10%]
```

ここでは、オリジナルの実メモリー・サイズ 4.25 GB は、2.75 GB の物理メモリー・サイズと拡張係数 1.55 により達成されます。 この構成により、 CPU 使用は

0.39 物理プロセッサー分 (最大能力の 10 %) だけ増加する可能性があります。

Active Memory Expansion 推奨

このセクションでは、ワークロードに対して作成された Active Memory Expansion 構成の推奨が表示されます。最適な構成としては、AME による CPU 使用率は 15% を超えないようにします。 このセクシ

ョンでは、推奨される構成に対して「AME CPU Usage」と「Memory Gain」に関する情報も表示します。前述のレポートでは、拡張係数 2.12 が推奨されています。 AME による CPU 使用のターゲットは、**-c** または **-C** オプションで変更可能です。

注**:** このセクションで作成されたすべての推奨値は、モニター期間中に実行したワークロードに基づいています。 このセクションにレポートされた AME による CPU 使用状況は推定値であり、変わる可能性があります。 **amepat** のモデリング出力は、異なるレベルのメモリー拡張において、ワークロード・スループットまたは応答時間がどのように変わるかを見積もるものではありません。 Active Memory Expansion を使用してワークロードを実稼働する前に、使用される構成を綿密にモニターしてそのワークロードのパフォーマンス目標が満たされていることを確認する必要があります。 **amepat** を AME が使用可能な環境で実行する場合、拡張メモリー・サイズに不足がある時は警告が出されます。

注**:** その出力は、以下のようになります。

```
WARNING: This LPAR currently has a memory deficit of 562 MB.
A memory deficit is caused by a memory expansion factor that is too
high for the current workload.  It is recommended that you reconfigure
the LPAR to eliminate this memory deficit.  Reconfiguring the LPAR
with one of the recommended configurations in the above table should
eliminate this memory deficit.

The recommended AME configuration for this workload is to configure
the LPAR with a memory size of ...
```

これらに関する詳細、およびその他の AME 計画ツールの使用に関する詳細は、**amepat** のマニュアル・ページを参照してください。

パフォーマンス・モニター

いくつかの AIX パフォーマンス・ツールを使用して、Active Memory Expansion 統計をモニターし、Active Memory Expansion に関する情報を収集できます。

以下のテーブルには、Active Memory Expansion 統計をモニターするのに使用可能な種々のツールとオプションを要約してあります。

ツール	オプション	説明
amepat	**-N**	全般的な CPU とメモリーの使用状況統計を提供します。また、圧縮統計、解凍統計、およびメモリー不足統計の他に、AME 圧縮/解凍アクティビティーのための CPU 使用率の表示も行います。
vmstat	**-c**	圧縮、解凍、および不足に関する統計を提供します。
lparstat	**-c**	AME 圧縮/解凍アクティビティーのための CPU 使用率の表示も行います。 メモリー不足情報も提供します。
svmon	**-O summary=ame**	圧縮されたページと圧縮されていないページに分けて、メモリー使用の要約を表示します。
topas		デフォルトの topas 画面では、このツールが AME 環境で実行された場合のメモリー圧縮統計が表示されます。

vmstat コマンド

vmstat コマンドを **–c** オプション指定で発行して、AME 統計を表示できます。

```
# vmstat –c 2 1

System configuration: lcpu=2 mem=1024MB tmem=512MB ent=0.40 mmode=dedicated-E
```

```
kthr               memory                        page          faults
 r   b     avm   fre    csz   cfr    dxm   ci   co  pi  po  in   sy   cs
 0   0  309635  2163  43332   943  26267  174  386   0   0  93  351  339

cpu
us sy id wa    pc    ec
2  3 89  7  0.02   5.3
```

上記の出力では、以下のメモリー圧縮統計が表示されています。

- LPAR の拡張メモリー・サイズ **mem** は 1024 MB です。
- LPAR の実メモリー・サイズ **tmem** は 512 MB です。
- LPAR のメモリー・モード **mmode** は、Active Memory Sharing が使用不可で、Active Memory Expansion が使用可能です。
- 圧縮プール・サイズ **csz** は 43332 (4K ページ数) です。
- 圧縮プール内の空きメモリー量 **cfr** は 943 (4K ページ数) です。
- 拡張メモリー不足のサイズ **dxm** は 26267 (4K ページ数) です。
- 圧縮操作回数または圧縮プールへのページアウト回数 **co** は、386 回/秒です。
- 圧縮プールからの解凍操作回数またはページイン回数 **ci** は、174 回/秒です。

lparstat コマンド

lparstat コマンドを **–c** オプション指定で発行して、AME 統計を表示できます。

```
# lparstat -c 2 5

System configuration: type=Shared mode=Uncapped mmode=Ded-E smt=On
lcpu=2 mem=1024MB tmem=512MB psize=14 ent=0.40

%user  %sys  %wait  %idle physc %entc  lbusy   app  vcsw phint  %xcpu    dxm
----- ----- ------ ------ ----- ----- ------   --- ----- ----- ------ ------
 45.6  51.3    0.2    2.8  0.95 236.5   62.6 11.82  7024     2    5.8    165
 46.1  50.9    0.1    2.8  0.98 243.8   64.5 11.80  7088     7    6.0    162
 46.8  50.9    0.3    2.1  0.96 241.1   69.6 11.30  5413     6   19.4    163
 49.1  50.7    0.0    0.3  0.99 247.3   60.8 10.82   636     4    8.6    152
 49.3  50.5    0.0    0.3  1.00 248.9   56.7 11.47   659     1    0.3    153
```

この出力例では、以下のメモリー圧縮統計が表示されています。

- LPAR のメモリー・モード **mmode** は、Active Memory Sharing が使用不可で、AME が使用可能です。
- LPAR の拡張メモリー・サイズ **mem** は 1024 MB です。
- LPAR の実メモリー・サイズ **tmem** は 512 MB です。
- Active Memory Expansion アクティビティー用に使用された CPU 使用率は、**%xcpu** に表示されています。
- 拡張メモリー不足のサイズ **dxm** は、メガバイト単位で表示されます。

topas コマンド

Active Memory Expansion が使用可能状態の LPAR の **topas** メインパネルでは、サブセクション **AME** の下にメモリー圧縮統計を自動的に表示します。

```
Topas Monitor for host:    proc7               EVENTS/QUEUES    FILE/TTY
Mon Dec 14 16:30:50 2009   Interval:  2        Cswitch    1240  Readch  43.2M
                                               Syscall  110.8K  Writech 102.5K
CPU  User%  Kern%  Wait%  Idle%  Physc   Entc   Reads     12594  Rawin     0
ALL   49.1   50.7    0.0    0.3   1.00  249.7   Writes      515  Ttyout   388
```

```
                                                  Forks       218  Igets      0
Network  KBPS   I-Pack  O-Pack   KB-In  KB-Out   Execs       218  Namei    5898
Total     1.2      7.5     1.0     0.9     0.3   Runqueue    1.0  Dirblk      0
                                                  Waitqueue   0.0
Disk    Busy%     KBPS     TPS KB-Read KB-Writ                    MEMORY
Total     0.0      0.0     0.0     0.0     0.0   PAGING           Real,MB   1024
                                                  Faults    53184  % Comp     85
FileSystem        KBPS     TPS KB-Read KB-Writ   Steals        0  % Noncomp   0
Total             75.4K   21.1K  75.3K  95.4    PgspIn        0  % Client    0
                                                  PgspOut       0
WLM-Class (Active)     CPU%    Mem%  Blk-I/O%     PageIn        0  PAGING SPACE
System                    0      61         0   PageOut       0  Size,MB    512
Default                   0       4         0   Sios          0  % Used      1
                                                                 % Free     99
Name            PID CPU% PgSp Class              AME
inetd        364682  3.5  0.5 wpar1              TMEM,MB     512  WPAR Activ   1
xmtopas      622740  0.4  0.7 wpar1              CMEM,MB     114  WPAR Total   1
topas        413712  0.1  1.5 System             EF[T/A] 2.0/1.5  Press: "h"-help
random       204934  0.1  0.1 System             CI:5.5  CO:0.0          "q"-quit
```

上記の出力例では、以下のメモリー圧縮統計が表示されています。

- LPAR の実メモリー・サイズ **TMEM,MB** は 512 MB です。
- LPAR の圧縮プール・サイズ **CMEM,MB** は 114 MB です。
- **EF[T/A]** – ターゲット拡張係数は 2.0 で、達成された拡張係数は 1.5 です。
- 圧縮発生率 **co**/秒と解凍発生率 **ci**/秒は、それぞれ 0.0 と 5.5 ページ/秒です。

svmon コマンド

svmon ツールでは、LPAR 上での AME 使用詳細を表示できます。

```
# svmon -G -O summary=ame,pgsz=on,unit=MB
Unit: MB
--------------------------------------------------------------------------------
               size       inuse        free   pin     virtual  available mmode
memory      1024.00      607.54      144.11   306.29  559.75    136.61   Ded-E
 ucomprsd         -      387.55           -
 comprsd          -      219.98           -
pg space     512.00        5.08

               work        pers        clnt       other
pin          213.34           0           0        28.9
in use       534.12           0        9.42
 ucomprsd    314.13
 comprsd     219.98

PageSize   PoolSize       inuse        pgsp         pin     virtual   ucomprsd
s    4 KB         -      543.54        5.02      242.27      560.59     323.55
L   16 MB         4           0           0        64.0           0          0
--------------------------------------------------------------------------------
True Memory: 512.00

              CurSz  %Cur       TgtSz        %Tgt       MaxSz   %Max   CRatio
ucomprsd     405.93  79.28      168.38       32.89      -          -    -
comprsd      106.07  20.72      343.62       67.11      159.59   31.17  2.51

               txf         cxf         dxf         dxm
AME           2.00        1.46        0.54      274.21
```

上記の出力例では、以下のメモリー圧縮統計が表示されています。

- LPAR のメモリー・モード **mmode** は、Active Memory Sharing が使用不可で、AME が使用可能です。

- 使用中の合計サイズ (**memory_inuse**) 607.54 MB の内で、非圧縮ページ (**ucomprsd_inuse**) は 387.55 MB であり、圧縮ページ数 (**comprsd_inuse**) は残りの 219.98 MB となります。
- 使用中の合計作業中ページ (**inuse_work**) 534.12 MB の内で、非圧縮ページ (**ucomprsd_work**) は 314.13 MB であり、圧縮ページ (**comprsd_work**) は 219.98 MB となります。
- 4K ページ・サイズ・プールでの使用中ページ (**4KB_inuse**) の合計 543.54 MB の内で、非圧縮ページ (**4KB_ucomprsd**) は 323.55 MB となります。
- LPAR の拡張メモリー・サイズ **memory_size** は 1024 MB です。
- LPAR の実メモリー・サイズ **True Memory** は 512 MB です。
- 非圧縮プールの現行サイズ (**ucomprsd_CurSz**) は 405.93 MB (LPAR の合計実メモリー・サイズの 79.28%) です。
- 圧縮プールの現行サイズ (**comprsd_CurSz**) は 106.07 MB (LPAR の合計実メモリー・サイズの 20.72%) です。
- ターゲット・メモリー拡張係数 (**txf**) 2.00 を達成するのに必要な圧縮プールのターゲット・サイズ (**comprsd_TgtSz**) は、343.62 MB (LPAR の合計実メモリー・サイズの 67.11%) です。
- 上記出力例のケースでの非圧縮プールのサイズ (**ucomprsd_TgtSz**) は、168.38 MB (LPAR の合計実メモリー・サイズの 32.89%) になります。
- 圧縮プールの最大サイズ (**comprsd_MaxSz**) は 159.59 MB (LPAR の合計実メモリー・サイズの 31.17%) です。
- 現行の圧縮率 (**CRatio**) は 2.51 であり、達成された現行の拡張係数 (**cxf**) は 1.46 です。
- 拡張メモリー不足量 (**dxm**) は 274.21 MB であり、不足拡張係数 (**dxf**) は 0.54 です。

–O summary=longame オプションを使用すると、以下のようにメモリー圧縮詳細が表示されます。

```
# svmon -G -O summary=longame,unit=MB
Unit: MB
                                    Active Memory Expansion
--------------------------------------------------------------------
 Size     Inuse     Free   DXMSz  UCMInuse  CMInuse   TMSz    TMFr
1024.00  607.91  142.82  274.96   388.56    219.35   512.00  17.4

CPSz  CPFr  txf  cxf   CR
106.07 18.7 2.00 1.46   2.50
```

この出力例では、以下のメモリー圧縮統計が表示されています。

- 合計拡張メモリー・サイズ (**Size**) 1024.00 MB の内で、 607.91 MB が使用中 (**Inuse**) であり、142.82 MB が空き状態 (**Free**) です。 拡張メモリー・サイズ内の不足 (**DXMSz**) は 274.96 MB です。
- 合計使用中メモリー (**Inuse**) 607.91 MB の内で、非圧縮のページ (**UCMInuse**) は 388.56 MB であり、圧縮ページ (**CMInuse**) は残りの 219.35 MB となります。
- 実メモリー・サイズ (**TMSz**) 512.00 MB の内で、空き状態の実メモリー (**TMFr**) の 17.4 MB だけが使用可能です。
- 圧縮プール・サイズ (**CPSz**) 106.07 MB の内で、空き状態のメモリー (**CPFr**) の 18.7 MB だけが圧縮プールで使用可能です。
- ターゲット拡張係数 (**txf**) が 2.00 であるのに対し、達成された現行拡張係数 (**cxf**) が 1.46 です。
- 圧縮率 (CR) は 2.50 です。

関連情報:

vmo コマンド

アプリケーションのチューニング

プログラムのパフォーマンスを改善するために多くの労力を費やす前に、このセクションの手法を使用して、パフォーマンスをどの程度改善できるのかを判断し、最適化およびチューニングが最も有効であるプログラムの領域を検出してください。

一般的に、最適化プロセスには、以下に示すような幾つかのステップが含まれます。

- チューニングによっては、ステートメントおよび式の順序の変更などによる、ソース・コードの変更が含まれます。 この手法は、ハンド・チューニング と呼ばれます。
- FORTRAN および C プログラムの場合、ソース・コードをコンパイル前にチューニングまたはその他の方法で変換する最適化プリプロセッサーが使用可能です。 このようなプリプロセッサーの出力は、最適化済みの FORTRAN または C ソース・コードです。
- FORTRAN または C++ コンパイラーはソース・コードを中間言語に変換します。
- コード・ジェネレーターは、中間コードをマシン言語に変換します。コード・ジェネレーターは、選択されたコンパイラー・オプションに従って、最終的な実行可能コードを最適化して、速度を増加することができます。 最初にハンド・チューニングまたはプリプロセスを行うことによって、このステップで実行される最適化の量を増加することができます。

速度の増加は、以下の 2 つの要因の影響を受けます。

- プログラムの各部分に適用される最適化の量
- 実行時にプログラムのそれらの部分が使用される頻度

単一のルーチンが作業の大半を実行する場合には、そのルーチンの速度を増加すれば、プログラムの速度を大幅に増加することができます。一方、そのルーチンがほとんど呼び出されない場合、および実行に長い時間がかからない場合には、パフォーマンス全体はそれほど改善されません。 パフォーマンスの手法およびデータを評価する場合に、この点に注意して、実行する作業に関して最も価値のある手法を集中的に使用してください。

これらの手法の詳細については、「*Optimization and Tuning Guide for XL Fortran, XL C and XL C++*」を参照してください。 追加のヒントについては、 96 ページの『効率的なプログラム設計とインプリメンテーション』も参照してください。

コンパイラーの最適化手法

コンパイラーの最適化には幾つかの手法があります。

ソース・コード・チューニングの 3 つの主な領域を以下に示します。

- 最適化コンパイラーおよびシステム・アーキテクチャーを利用するプログラミング手法。
- 基本的な線形代数サブルーチンのライブラリーである BLAS。 数値指向のプログラムがある場合、これらのサブルーチンはパフォーマンスをかなり改善することができます。 BLAS の拡張機能は ESSL (工学科学サブルーチン・ライブラリー) です。 ESSL には、BLAS ライブラリーのサブセットの他に、化学、工学、および物理学のための高性能の数学ルーチンが含まれています。 SMP マシン用の並列 ESSL (PESSL) があります。
- コンパイラー・オプション、およびサード・パーティーのベンダーから入手できる KAP と VAST などのプリプロセッサーの使用。

このようなソース・コード・チューニング手法に加えて、 **fdpr** プログラムは、オブジェクト・コードを再構成します。 **fdpr** プログラムの説明は、 130 ページの『fdpr プログラムによる実行可能プログラムの再構造化』にあります。

最適化を使用するコンパイル

最適なパフォーマンスを達成するプログラムを生成する最初のステップは、コンパイラーに組み込まれている基本的な最適化フィーチャーを利用することです。

最適化を指定してコンパイルすることによって、プログラム・チューニングの結果であるスピードアップを増加することができ、ある種のチューニングを行う必要がなくなります。

推奨事項

最適化のための以下のガイドラインに従ってください。

- コンパイルする任意の実動レベルの FORTRAN、C、または C++ プログラムに **-O2** または **-O3 -qstrict** を使用します。 高性能 FORTRAN (HPF) プログラムの場合には、**-qstrict** オプションを使用しないでください。
- ホット・スポットがループであるか、または配列言語であるプログラムの場合には、**-qhot** オプションを使用してください。 HPF プログラムには常に **-qhot** オプションを使用してください。
- コンパイル時間が特に問題にならない場合には、開発サイクルの最後に近づいてから **-qipa** オプションを使用してください。

-qipa オプションは、プロシージャー間分析 として知られる最適化のクラスを活動化またはカスタマイズします。 **-qipa** オプションには幾つかのサブオプションがあります。サブオプションの説明はコンパイラーのマニュアルにあります。 このオプションは、次の 2 つの方法で使用することができます。

- 最初の方法は、コンパイルとリンクの両方のステップで、**-qipa** オプションを使用してコンパイルします。 コンパイル時に、コンパイラーはプロシージャー間分析情報を `.o` ファイルに保管します。 リンク時に、**-qipa** オプションによって、アプリケーション全体の完全な再コンパイルが実行されます。
- 2 番目の方法は、**-p**/**-pg** オプションを指定して (**-qipa** を指定して、または指定しないで)、プロファイルを作成するためにプログラムをコンパイルし、データの典型的なセットに対して実行します。 次に、結果のデータを、**-qipa** を指定した後続のコンパイルへの入力として使用して、コンパイラーが、最も頻繁に使用される 2 番目以降のプログラムの最適化に集中できるようにします。

-O4 を使用することは、**-O3 -qipa** を使用することと同じで、そのプラットフォームに最適なアーキテクチャーとチューニング・オプションが自動生成されます。 **-O5** フラグは、**-O4** と類似していますが、**-qipa**= level = 2 である点が異なります。

コンパイラーの最適化の使用には、以下の利点があります。

分岐の最適化
: プログラム・コードを再配置して、分岐のロジックを最小化し、物理的に離れているコードのブロックを結合します。

コードの移動
: ループ内の計算で使用される変数がループ内で変更されない場合、計算をループの外側で実行し、結果をループ内で使用することができます。

共通副次式の除去
: 共通式では、後続の式で同じ値が再計算されます。 以前の値を使用することによって、重複する式を除去することができます。

定数の伝搬
: 式内で使用される定数が結合され、新規の定数が生成されます。整数型と浮動小数点型の間のいくつかの暗黙の型変換が実行されます。

デッド・コードの除去
: 到達できないコードまたは結果が以後使用されないコードを除去します。

デッド・ストアの除去
: 保管された値が再び参照されない場合の保管内容を除去します。 例えば、同じロケーションへの 2 回の保管の間にロードがない場合、最初の保管内容は不要なので除去されます。

グローバル・レジスターの割り当て
: 変数および式を使用可能なハードウェア・レジスターへ、「グラフ・カラー化」アルゴリズムを使用して割り当てます。

インライン化
: 関数呼び出しを実際のプログラム・コードで置換します。

命令のスケジューリング
: 実行時間を最小化するために、命令の順序を変更します。

プロシージャー間分析
: 関数呼び出し間の関係を明らかにし、より単純な最適化では除去できないロード、保管、および計算を除去します。

変化しない **IF** コードの移動 **(**スイッチングなし**)**
: ループから変化しない分岐コードを除去して、他の最適化の機会を増やします。

プロファイル主導のフィードバック
: サンプル・プログラムの実行の結果が、条件付き分岐近くで、また、頻繁に実行されるコード・セクションで、最適化の改善に使用されます。

再関連付け
: 共通式の除去用のより多くの候補を生成するために、配列添え字式内の一連の計算を再配置します。

保管の移動
: 保管命令をループから移動します。

強度の削減
: 非効率的な命令を、より効率的な命令で置換します。 例えば、配列の添え字付けでは、乗算命令を加算命令で置換します。

値の番号付け
: 定数の伝搬、式の除去、および複数の命令の単一の命令への変換を含みます。

最適化なしでコンパイルする必要がある場合

シンボリック・デバッガーを使用してデバッグするプログラムには、**-g** オプションの使用の有無に関係なく、**-O** オプションを使用しないでください。 ただし、最適化は HPF プログラムにとって非常に重要なので、デバッグ中でも **-O3 -qhot** を使用してください。

最適化プログラムは、アセンブラー言語命令を再配置するので、個々の命令を 1 行のソース・コードにマップすることは困難になります。 **-g** オプションを使用してコンパイルすると、この再配置によって、シンボリック・デバッガーを使用した場合に、ソース・レベル・ステートメントが正しくない順序で実行されているかのように見える可能性があります。

いずれかの **-O** オプションを使用してコンパイルした時に、プログラムが正しくない結果を生成した場合には、プログラムのプロシージャー参照で、変数に意図せずに別名を割り当てていないかどうかを調べてください。

特定のハードウェア・プラットフォームのためのコンパイル

特定のハードウェア・プラットフォーム用にコンパイルするには、幾つかの考慮すべき項目があります。

システムは幾つかのタイプのプロセッサーを使用することができます。 **-qarch** および **-qtune** オプションを使用することによって、これらのプロセッサーの特殊な命令および特定の強度に関してプログラムを最適化できます。

推奨事項

特定のハードウェア・プラットフォーム用のコンパイルについては、以下のガイドラインに従ってください。

- プログラムを単一のシステムでのみ、または同じプロセッサー・タイプのシステムのグループでのみ実行する場合には、**-qarch** オプションを使用して、プロセッサー・タイプを指定してください。
- プログラムを複数の異なるプロセッサー・タイプを持つシステムで実行し、かつ、最も重要なプロセッサー・タイプを 1 つ識別できる場合には、該当する **-qarch** と **-qtune** の設定値を使用してください。XL FORTRAN および XL HPF ユーザーは、これらの設定値を対話式に選択するために、**xxlf** および **xxlhpf** コマンドを使用することができます。
- プログラムをすべての範囲のプロセッサー・インプリメンテーションで実行する予定であり、特定のプロセッサー・タイプを主に使用することがない場合には、**-qarch** も **-qtune** も使用しないでください。

浮動小数点パフォーマンスのためのコンパイル

浮動小数点指向のプログラムのパフォーマンスを改善するために、いくつかのデフォルトの浮動小数点オプションを変更することができます。

このようなオプションのいくつかは、浮動小数点標準への準拠に影響を与えます。これらのオプションを使用すると、計算の結果が変更される可能性がありますが、多くの場合、結果として正確度は増加します。

推奨事項

以下のガイドラインに従ってください。

- POWER family・プラットフォームと POWER2 プラットフォーム上の単精度プログラムの場合、これらの浮動小数点オプションを使用することによって、正確度を保持しながら、パフォーマンスを改善することができます。

  ```
  -qfloat=fltint:rsqrt:hssngl
  ```

  単精度プログラムがメモリー指向でない場合 (例えば、使用可能なキャッシュ・スペースより多くのデータにアクセスしない場合)、以下のものを使用することによって、少なくともこれまで以上のパフォーマンスと精度を取得することができます。

  ```
  -qfloat=fltint:rsqrt -qautodbl=dblpad4
  ```

  単精度変数を含んでいないプログラムの場合には、**-qfloat=rsqrt:fltint** のみを使用してください。**-qstrict** を指定しない **-O3** は、自動的に **-qfloat=rsqrt:fltint** を設定することに注意してください。
- 単精度プログラムは一般的に倍精度プログラムより効率的なので、デフォルトの REAL 値を REAL(8) にプロモートすると、パフォーマンスが低下する可能性があります。次の **-qfloat** サブオプションを使用してください。

キャッシュ・サイズの指定

プログラムを単一のマシンまたは構成だけで実行する予定であれば、 **-qcache** オプションを使用することによって、コンパイラーに、そのマシンのメモリー・レイアウトに合うようにプログラムをチューニングすることができます。

-qcache オプションを効果的に使用するには、**-qhot** オプションも指定する必要があります。 **-qhot** オプションは **-qcache** 情報を使用して、該当するメモリー管理の最適化を判別します。

キャッシュには、データ、命令、およびそれらの結合の 3 つのタイプがあります。 モデルは一般的に、データと命令の両方のキャッシュを備えたモデルと、単一のデータ/命令結合キャッシュを備えたモデルの 2 つのカテゴリーに分けられます。 TYPE サブオプションによって、 **-qcache** オプションが参照するキャッシュのタイプを識別することができます。

-qcache オプションは、サイズを識別し、モデルのレベル 2 キャッシュと変換索引バッファー (TLB) の結合順序を設定するために使用することもできます。この機構は、最近参照されたメモリーのページを見つけるために使用されるテーブルです。 多くの場合、プログラムが 512 KB より大きなデータ・スペースを使用していない限り、 TLB に関する **-qcache** エントリーを指定する必要はありません。

実行時のシステム負荷によっては、SIZE 属性の設定値を低くすると、パフォーマンスが改善される場合があります。

プロシージャー・コールのインライン展開

インライン化 には、参照されるプロシージャーの、参照元のコードへのコピーが含まれます。 これは、インライン化されたルーチンの呼び出しのオーバーヘッドを除去し、最適化プログラムによるインライン化されたルーチン内の他の最適化の実行を可能にします。

FORTRAN および C プログラムの場合、**-Q** オプションを (**-O2** または **-O3** と共に) 指定すれば、プロシージャーを参照点にインラインで組み込むことができます。

インライン化によって、パフォーマンスが改善されるプログラムと、パフォーマンスが低下するプログラムがあります。 インライン化によってプログラムのパフォーマンスが低下する理由は、大きなコード・サイズの結果として、キャッシュ・ミスやページ・フォールトが増加するため、あるいは、いくつかの結合ルーチン内にすべてのローカル変数を収容するための十分なレジスターがないためです。

-Q オプションを使用する場合には、常に、**-O3** および **-Q** を指定してコンパイルしたプログラムのバージョンのパフォーマンスを、**-O3** のみを指定してコンパイルした場合と比較してください。 **-Q** を指定してコンパイルしたプログラムのパフォーマンスは、劇的に改善される場合、劇的に低下する場合、あるいは、ほとんどまたはまったく変らない場合があります。

コンパイラーは、プロシージャーのサイズに基づいて、インライン化するかどうかを判断します。 インライン化に関する他の基準を使用して、アプリケーションのパフォーマンスを改善することができることもあります。 一般的な実行では参照される可能性が少ないプロシージャー (例えば、エラー処理およびデバッグ・プロシージャー) の場合、 **-Q-names** オプションを使用して、インライン化を選択的に使用不可にします。 ホット・スポット内で参照されるプロシージャーに関しては、 **-Q+names** オプションを指定して、このようなプロシージャーが常にインライン化されるようにします。

ダイナミック・リンクおよび静的リンクを使用する時期

オペレーティング・システムには、ダイナミック・リンク共用ライブラリーを作成して使用する機能が備わっています。 ダイナミック・リンクを使用すると、ユーザー・コード内で参照されていて共用ライブラリ

ーに定義されている外部シンボルは、ロード時にローダーによって解決されます。 共用ライブラリーを使用するプログラムをコンパイルすると、共用ライブラリーはデフォルトでプログラムに動的にリンクされます。

共用ライブラリーの基礎にある考え方は、共用されるルーチンのコピーを 1 つだけ用意して、この共通コピーを各自の共用ライブラリー・セグメントに維持することです。このような共通ルーチンによって、実行可能プログラムのサイズを大幅に縮小できるので、ディスク・スペースを節約できます。

ダイナミック・リンクを使用することによって、プログラムのサイズを縮小できますが、通常は、パフォーマンス上のトレードオフがあります。 共用ライブラリーのコードは、ディスク上の実行可能イメージには存在せずに、別個のライブラリー・ファイル内に保持されます。 共用コードは、共用ライブラリー・セグメント内のメモリーに一度ロードされ、それを参照するすべてのプロセスによって共用されます。 したがって、幾つかの並行して実行中のアプリケーション (または同じアプリケーションのコピー) が共用ライブラリー内のプロシージャーを使用する場合には、ダイナミック・リンク・ライブラリーによって、プログラムが使用する仮想記憶域の量が減らされます。 特定のシステムに保管されている幾つかの異なるアプリケーションがライブラリーを共用する場合には、プログラムに必要なディスク・スペースの量も減らされます。 共用ライブラリーのその他の利点を以下に示します。

- 共用ライブラリー・コードが既にメモリー内にある場合があるので、ロード時間が減らされる可能性があります。
- いくつかのアプリケーションまたはアプリケーションのコピーが使用中である共用ライブラリー・コードは、単一のアプリケーションのみが使用中のコードと比較すると、オペレーティング・システムがページアウトする可能性が低いので、実行時パフォーマンスを改善することができます。 結果として、ページ・フォールトの発生も少なくなります。
- ルーチンはアプリケーションに静的に結合されずに、アプリケーションのロード時に動的に結合されます。 これによって、アプリケーションは、共用ライブラリーに対する変更を、再コンパイルまたは再バインドなしで、自動的に継承することができます。

ダイナミック・リンクの欠点を以下に示します。

- パフォーマンスの観点から言えば、共用セグメントにアクセスするために、実行可能プログラムに「グルー・コード」が必要です。 共用ライブラリー・ルーチンの参照には、参照当たり約 8 マシン・サイクルのパフォーマンス・コストがかかります。 共用ライブラリーを使用するプログラムは、通常、静的にリンクしたライブラリーを使用するプログラムより遅くなります。
- より複雑な影響は、「参照の局所性」の減少です。 プログラムがライブラリー内のわずかなルーチンのみを使用し、これらのルーチンがライブラリーの仮想アドレス・スペース内で広範囲に分散している場合があります。 したがって、これらのすべてのルーチンが実行可能プログラムに直接結合されている場合と比較して、すべてのルーチンにアクセスするためにタッチする必要がある合計ページ数は、かなり多くなります。 この状況に関する 1 つの影響は、ユーザーがこのようなルーチンの唯一のユーザーである場合には、すべてのルーチンを実メモリーに置くために、より多くのページ・フォールトを経験することになります。 さらに、より多くのページが操作の対象になるので、命令の変換索引バッファー (TLB) ミスが発生する可能性も高くなります。
- プログラムがライブラリー内の限られた数のプロシージャーを参照する場合に、参照対象のプロシージャーを含むライブラリーの各ページを、個別に実メモリーにページインしなければなりません。 プロシージャーが小さいために、静的リンクを使用すれば、異なるライブラリー・ページにある複数のプロシージャーを単一のページにリンクできる場合には、ダイナミック・リンクを使用するとページングが増加するので、パフォーマンスが低下する可能性があります。
- ダイナミック・リンク・プログラムは、互換性のあるライブラリーの所有に依存します。ライブラリーが変更された場合 (例えば、新規コンパイラー・リリースでライブラリーが変更された場合)、ライブラ

リーの新規バージョンとの互換性のために、アプリケーションを作り直さなければならない可能性があります。 ライブラリーがシステムから除去された場合には、そのライブラリーを使用しているプログラムは機能しなくなります。

静的にリンクされたプログラムでは、すべてのコードが単一の実行可能モジュールに含まれています。 ライブラリーのプロシージャーはプログラムに静的にリンクされるので、ライブラリー参照はより効率的です。 静的リンクは、プログラムのファイル・サイズを増やし、他のアプリケーションまたはアプリケーションの他のコピーがシステム上で実行中の場合には、メモリー内のコード・サイズを増やす可能性があります。

cc コマンドは、デフォルトで、共用ライブラリー・オプションをとります。プログラムをコンパイルする時にこのデフォルトをオーバーライドして、静的リンク・オブジェクト・ファイルを作成するには、以下のように **-bnso** オプションを使用してください。

```
cc xxx.c -o xxx.noshr -O -bnso -bI:/lib/syscalls.exp
```

このオプションによって、リンカーは、プログラムが参照するライブラリーのプロシージャーを、プログラムのオブジェクト・ファイルに置きます。 `/lib/syscalIs.exp` ファイルには、システムからプログラムにインポートする必要があるシステム・ルーチンの名前が入っています。 静的リンクの場合、このファイルを指定する必要があります。 ダイナミック・リンクの場合、このファイルに名前が入っているルーチンは、`libc.a` によって自動的にインポートされるので、ダイナミック・リンク時に、このファイルを指定する必要はありません。上記のオプションの詳細については、 447 ページの『ld コマンドの効率的な使用法』および **Id** コマンドを参照してください。

非共用ライブラリーがパフォーマンスを支援するかどうかの判別**:**

アプリケーションが共用ライブラリーの方法によってパフォーマンスが低下しているかどうかを判別する 1 つの方法は、非共用オプションを使用して実行可能プログラムを再コンパイルすることです。

パフォーマンスがかなりよくなれば、パフォーマンス向上のために、共用ライブラリーの他の利点を犠牲にすることを考える必要があるかもしれません。 ただし、実際の環境でパフォーマンスを測定してください。 プログラムは、負荷が軽いマシンの単一インスタンスとして非共用で結合されている場合、実行が速くなります。 同じプログラムを多くのユーザーが同時に使用する場合には、実メモリーの使用率が増加し、ワークロード全体の実行は遅くなります。

プリロードされた共用ライブラリー

LDR_PRELOAD および *LDR_PRELOAD64* 環境変数は、プロセスで共用ライブラリーをプリロードすることを可能にします。 *LDR_PRELOAD* 環境変数は 32 ビット・プロセス用、*LDR_PRELOAD64* 環境変数は 64 ビット・プロセス用です。

シンボルの解決では、環境変数にリストされたプリロード・ライブラリーが、最初にすべてのインポートされたシンボルごとに検索され、これらのライブラリーで検出されない場合のみ、通常の検索が使用されます。 プリロードされたライブラリーからのシンボルの先取りは、AIX のデフォルト・リンクとランタイム・リンクの両方で有効です。 据え置きシンボル解決は未変更です。

これらの環境変数について詳しくは、 468 ページの『各種チューナブル・パラメーター』を参照してください。

大きなプログラムのページングを減らすためのリンク順序の指定

プログラム・コンパイルのリンケージ・フェーズ時に、リンカーは、参照の局所性の改善を試みて、プログラム・ユニットを再配置します。

例えば、プロシージャーが別のプロシージャーを参照する場合、リンカーは、ロード・モジュール内で 2 つのプロシージャーを隣接させて、両方のプロシージャーが仮想メモリーの同じページに収まるようにします。 これにより、ページングのオーバーヘッドを減らすことができます。最初のプロシージャーが最初に参照され、このプロシージャーを含むページが実メモリーに持ってこられた場合、2 番目のプロシージャーは、追加のページング・オーバーヘッドなしで使用可能です。

プログラムのコードのページに関して、過度なページングが発生する非常に大きなプログラムの場合、リンカーで特定のリンク順序を指定することができます。 これを実行するには、制御セクションをリンクしたい順序に配置し、**-bnoobjreorder** オプションを使用して、リンカーによる順序の変更を防ぎます。 制御セクション (CSECT) は、XCOFF オブジェクト・モジュール内のコードまたはデータの、最も小さい置換可能なユニットです。 詳細については、ファイル参照を参照してください。

ただし、リンク順序の指定には、多くのリスクが伴います。 すべてのリンクの順序の変更の後に完全なパフォーマンス・テストを常に実行し、新規のリンク順序が、リンカーによって選択されるリンク順序と比較して、プログラムにとってよりよい結果をもたらすことを確認する必要があります。 独自のリンク順序の確立を決定する前に、以下の点について考慮してください。

- プログラム内のすべての CSECT に関してリンク順序を決定する必要があります。 CSECT は、リンクしたい順序でリンカーに提示する必要があります。 大きなプログラムでは、このような順序付けの作業には相当な労力がかかり、エラーが発生しやすくなります。
- プログラムの開発中に認められたパフォーマンス上の利点は、後からパフォーマンス上の欠点になる可能性があります。なぜなら、コード・サイズの変更によって、以前は 1 ページ内に一緒に置かれていた CSECT が、別個のページに分割される場合があるからです。
- 順序の変更は、命令のキャッシュ線の衝突の頻度を変更する可能性があります。命令キャッシュまたは両方向セットの結合であるデータと命令の結合キャッシュを備えたインプリメンテーションでは、プログラム・コードのすべての行は、キャッシュの 2 つの線のいずれかにしか保管できません。 3 つ以上の短い、相互に依存するプロシージャーのキャッシュ合致クラスが同じである場合、命令とキャッシュのスラッシングによってパフォーマンスが低下する可能性があります。 順序の変更によって、以前には発生しなかったキャッシュ線の衝突が発生する可能性があります。 順序の変更によって、**-bnoobjreorder** を指定しなかった場合に発生するキャッシュ線の衝突が除去される可能性もあります。

プログラムのリンク順序のチューニングを試みる場合には常に、他のプログラムによる実記憶装置とメモリーの合計の使用率が予想される作業環境に似たシステムで、パフォーマンスをテストしてください。 実行中のタスクが少ない、負荷が軽いシステムで有効なリンク順序は、負荷が重いシステムではページ・スラッシングを引き起こす可能性があります。

BLAS および ESSL ライブラリーの呼び出し

基本線形代数サブルーチン (BLAS) は、行列間、行列とベクトル間、およびベクトル間の演算における線形代数方程式に関して、高いレベルのパフォーマンスを提供します。 Engineering and Scientific Subroutine Library (ESSL) は、より包括的なサブルーチン・セットを含みます。すべてのサブルーチンは POWER プロセッサー・ベース ファミリー、POWER2、および PowerPC アーキテクチャー用にチューニングされます。

BLAS および ESSL サブルーチンによって、多くの算術演算をチューニングするための労力を省きながら、ハンド・チューニングによって取得できるものより、またはハンド・コーディングした算術演算の自動最適化によって取得できるものより、優れたパフォーマンスを提供することができます。
FORTRAN、C、および C++ プログラムで、両方のライブラリーから関数を呼び出すことができます。

BLAS ライブラリーは、基礎となるアーキテクチャーに関して高度にチューニングされた基本線形代数サブルーチンの集まりです。 BLAS サブセットは、オペレーティング・システムに備えられています (/lib/libblas.a)。

このライブラリーは、ユーザーが自分では達成できないレベルまでチューニングされているので、行列とベクトルの演算に関しては、このライブラリーを使用すべきです。

BLAS ルーチンは、FORTRAN プログラムから呼び出すように設計されていますが、 C プログラムでも使用することができます。 行列を参照する場合には、言語の差に注意する必要があります。 例えば、FORTRAN は配列を列優先順に保管しますが、C は行優先順を使用します。

/lib/libblas.a に存在する BLAS ライブラリーを組み込むには、コンパイラー・ステートメントで **-lblas** オプションを使用してください (**xlf -O prog.f -lblas**)。 C プログラムで BLAS を呼び出す場合には、FORTRAN ライブラリーに関する **-lxlf** オプションも組み込んでください (**cc -O prog.c -lblas -lxlf**)。

ESSL は、工学、化学および物理学の分野で使用されるさまざまな数学関数を含んでいる、さらに拡張されたライブラリーです。

BLAS または ESSL サブルーチンを使用する利点を以下に示します。

- BLAS および ESSL サブルーチン呼び出しは、置換対象の演算よりコーディングが容易です。
- BLAS および ESSL サブルーチンは、異なるプラットフォーム間で移植可能です。 サブルーチン名および呼び出しシーケンスは標準化されています。
- BLAS コードのパフォーマンスは、すべてのプラットフォーム上でよい可能性があります。 ルーチンの内部コーディングは通常、プラットフォームに固有なので、コードはアーキテクチャーのパフォーマンス特性に密接に結び付いています。

以下のプログラム例の 9 行の FORTRAN コードを、

```
do l=1,control
do j=1,control
        xmult=0.d0
        do k=1,control
                xmult=xmult+a(i,k)*a(k,j)
        end do
        b(i,j)=xmult
end do
end do
```

BLAS ルーチンを呼び出す以下の 1 行の FORTRAN コードで置換したところ、

```
call dgemm (`n','n',control,control,control,1,d0,a, control,a,1control,1.d0,b,control)
```

以下のパフォーマンスの改善が見られました。

配列のサイズ	**MULT** の経過時間	**BLAS** の経過時間	比率
101 x 101	.1200	.0500	2.40
201 x 201	.8900	.3700	2.41
301 x 301	16.4400	1.2300	13.37
401 x 401	65.3500	2.8700	22.77
501 x 501	170.4700	5.4100	31.51

この例は、行列の乗算を使用するプログラムが、パフォーマンスを改善するために、レベル 3 BLAS ルーチンを使用する方法を示しています。配列のサイズの増加に従って、改善も増加することに注意してください。

プロファイル指示フィードバック

PDF は、レジスターの割り当ての指示、命令のスケジューリング、および基本ブロックの再配置などの、プロシージャー・レベルの最適化をさらに実行するコンパイラー・オプションです。

PDF を使用するには、以下に挙げることを実行してください。

1. **-qpdf1** を使用してプログラム内のソース・ファイルをコンパイルします (関数 **main()** もコンパイルする必要があります)。 リンク・ステップ時に **-lpdf** オプションが必要です。使用する他のすべてのコンパイル・オプションも、ステップ 3 で使用する必要があります。
2. 一般的なデータ・セットを使用してプログラムを始めから終わりまで実行します。 プログラムは、終了時に、プロファイル作成情報を、*PDFDIR* 環境変数によって指定されるディレクトリー内、またはこの変数が設定されていない場合には、現在の作業ディレクトリー内の .__BLOCKS というファイルに記録します。異なるデータ・セットを使用してプログラムを複数回実行すれば、プロファイル作成情報が累積され、分岐が行われる頻度およびコードの各ブロックが実行される頻度に関する正確なカウントを入手することができます。 完成後のプログラムの通常の実行で使用される典型的なデータを使用することが重要です。
3. ステップ 1 と同じコンパイラー・オプションを使用してプログラムを再コンパイルしますが、 **-qpdf1** を **-qpdf2** に変更してください。 **-L** および **-l** がリンカー・オプションであり、この時点で変更できること、特に **-lpdf** オプションを省略できることに注意してください。 この 2 番目のコンパイルで、累積されたプロファイル作成情報が使用され、最適化のための細かいチューニングが行われます。結果のプログラムには、プロファイル作成のオーバーヘッドは含まれておらず、フルスピードで実行します。

PDFDIR ディレクトリーの管理に、以下の 2 つのコマンドを使用することができます。

resetpdf *pathname*
: pathname (パス名) ディレクトリーからすべてのプロファイル作成情報をクリアします (ただし、データ・ファイルは除去しません)。 pathname (パス名) を指定しない場合には、*PDFDIR* ディレクトリーから、 *PDFDIR* を設定していない場合には、現在のディレクトリーからクリアされます。アプリケーションを変更し、いくつかのファイルを再コンパイルした場合、これらのファイルのプロファイル作成情報は自動的にリセットされます。 再コンパイルされなかったプログラム部分に関する実行カウントに影響を与える可能性がある重要な変更を行った後には、 **resetpdf** コマンドを実行して、アプリケーション全体に関するプロファイル作成情報をリセットしてください。

cleanpdf *pathname*
: pathname (パス名) または *PDFDIR* または現在のディレクトリーから、すべてのプロファイル作成情報を除去します。 プロファイル情報を除去することによって、プログラムを変更してから PDF プロセスを再実行する場合の、実行時のオーバーヘッドが減らされます。 **-qpdf2** を指定してコンパイルした後に、このプログラムを実行してください。

fdpr コマンド

fdpr コマンドは、コンパイル済みの実行可能プログラム内のコードを再配置して、分岐パフォーマンスを改善し、ほとんど使用されないコードをプログラムのホット・スポットから移動し、その他のグローバルな最適化を実行します。

このコマンドは、多くの条件テストを含んでいる大きなプログラム、または複数のプロシージャーが散在している高度に構造化されたプログラムの場合に役立ちます。 **fdpr** コマンドの説明は、 130 ページの『fdpr プログラムによる実行可能プログラムの再構造化』にあります。

FORTRAN および C 用の最適化プリプロセッサー

1 組のプログラムをプリプロセッサーを使用してコンパイルし、プリプロセッサーを使用しないバージョンに関して、同じ最適化オプションを使用してコンパイルした場合と比較すると、パフォーマンス・テストは平均して 8 から 18% の範囲の改善を示します。

FORTRAN コンパイラー用の KAP および VAST プリプロセッサーは、POWER family、POWER2、および PowerPC 処理装置のリソースおよびメモリーの階層をよりよく使用できるように、FORTRAN ソース・コードを再構成することができます。 KAP プリプロセッサーのバージョンは、C プログラムのコードの再構成にも使用することができます。 プリプロセッサーは、メモリー管理の最適化、代数変換、インライン化、プロシージャー間分析、および FORTRAN または C アプリケーションのパフォーマンスを改善するその他の最適化を実行します。

KAP および VAST プリプロセッサーは、ソース・レベルのアルゴリズムを、コンパイラーの最適化能力を十分に活用できるアルゴリズムに変換しようとします。 このプリプロセッサーは、実行された変換を識別し、変換の実行を妨げるコードの領域を識別するリストも生成します。 プリプロセッサーはソース・コードを分析し、プログラムのパフォーマンスを改善する変換を実行します。

プリプロセッサーが実行するすべての変換は、ハンド・チューニングによっても実行することができます。ハンド・チューニングではなくプリプロセッサーを使用する利点を以下に示します。

- 多くの場合、プリプロセッサーは、プログラマーの多くの時間を費やすことなしに、ハンド・チューニングによる変換と同等またはより効率的に実行するプログラムを生成します。 プリプロセッサーを使用すれば、本書の別の個所で説明するアーキテクチャーまたはチューニング手法を完全に理解しないでもよくなります。
- プログラムによっては、適切なコマンド・ライン・プリプロセッサー・オプションを選択し、プログラムのソース・コードに数個のディレクティブを追加するだけで、高度に最適化されたコードを入手できます。 プリプロセッサーによって明白な改善が行われない場合には、プリプロセッサーのリストで、ソース・コードのどの領域が最適化を妨げているかを調べてください。
- プリプロセッサーが実行する変換の一部には、ソース・コードのかなりの拡張が含まれます。 このような拡張によってプログラムの効率が改善されますが、ハンド・チューニングによって拡張をインプリメントする場合、算法エラーおよびタイプミスの可能性が高くなり、ソース・コードの読み易さが減り、プログラムの保守がより困難になります。
- プリプロセッサーは、POWER family、POWER2、および PowerPC システムでは使用できないアーキテクチャーも含めて、特定のアーキテクチャーの構成用にチューニングされたコードを生成することができます。ソース・コードの単一のバージョンを維持し、さまざまな POWER family、POWER2、および PowerPC モデル用、または他のキャッシュ特性およびプロセッサー特性を持つマシン用にチューニングされた変換済みバージョンを生成することができます。
- プリプロセッサーは、しばしばハンド・チューニング済みのコードを改善することができます。 ハンド・チューニングによってプログラムを、プリプロセッサーによる効率のレベルまでチューニングすることは可能ですが、より複雑な一部の変換では、ハンド・チューニングによってコーディング・エラーが発生する可能性があります。

コードの最適化手法

メモリーとディスクの間の速度の差は、キャッシュとメモリーの間の差よりはるかに大きいので、メモリーの非効率な使用によるパフォーマンスの低下は、キャッシュの非効率な使用による低下よりはるかに大きいものです。

コードの最適化手法には、以下のものがあります。

- プログラムのコード作業セットを最小化するには、頻繁に実行するコードを一緒にパックして、頻繁に使用しないコードを分離します。 言い換えると、エラー処理コードの長いブロックをインラインに組み込まないようにし、頻繁に呼び出されるモジュールを呼び出し側のモジュールの隣りにロードします。
- データ作業セットを最小化するには、頻繁に使用するデータを一緒に集めて、ページへの不要な参照を回避します。 これは、**calloc()** サブルーチンの代わりに **malloc()** サブルーチンを使用し、データ構造体を使用直前に初期化して、割り当てられたスペースを、不要になった場合に確実に解放し、放棄することによって実行することができます。
- 固定されるストレージを最小化するには、固定されるコードを別個のロード・モジュールにパッケージ化します。固定されるコードの使用が必要かどうかを確認します。 特定のシステム構造体 (例えば、mbuf プール) がメモリー内に固定されます。このような構成を不必要に増加しないでください。
- コードをメモリー内に固定するために **plock()** サブルーチンのようなリアルタイム手法を使用することができます。優先順位は **setpri()** サブルーチンを使用して固定されます。

マップ・ファイル

マップ・ファイルの使用は、コードのもう 1 つの最適化手法です。

アプリケーションは、複数の read および write システム・コールを使用する代わりに、**shmat()** または **mmap()** システム・コールを使用して、アドレスによってファイルにアクセスすることができます。 システム・コールに関連するオーバーヘッドは常に存在するので、使用するコールは少ないほどよいのです。 **shmat()** または **mmap()** コールは、従来の **read()** または **write()** システム・コールに比べて、最高 50 倍までパフォーマンスを改善することができます。 **shmat()** サブルーチンを使用した場合、read または write システム・コールを使用した場合と同様に、ファイルがオープンされ、ファイル・ディスクリプター (*fd*) が戻されます。次に、**shmat()** コールはマップ・ファイルのアドレスを戻します。 複数の read システム・コールを使用する代わりに、エレメントをファイル内の後続のアドレスに等しく設定することによって、ファイルからマトリックスへの読み取りが行われます。

mmap() コールによって、セグメント境界を超えたメモリーのマッピングが可能になります。ユーザーは 10 を超える領域をメモリーにマップすることができます。 **mmap()** 関数は、メモリーの領域に関するページ・レベルの保護を提供します。 個々のページごとに、固有の読み取り権または書き込み権を持つことができるか、あるいは、アクセス許可セットを持つことができません。 **mmap()** コールによって、ファイルの 1 ページのみのマッピングが可能になります。

shmat() コールによって、マッピング対象のファイルがセグメントより大きい場合に、複数のセグメントのマッピングも可能になります。

以下のプログラム例は、read ステートメントを使用して、ファイルからの読み取りを行っています。

```
fd = open("myfile", O_RDONLY);
for (i=0;i<cols;i++) {
        for (j=0;j<rows;j++) {
                read(fd,&n,sizeof(char));
                *p++ = n;
        }
}
```

shmat() サブルーチンを使用すれば、read ステートメントを使用せずに、同じ結果を得ることができます。

```
fd = open("myfile", O_RDONLY);
nptr = (signed char *) shmat(fd,0,SHM_MAP | SHM_RDONLY);
for (i=0;i<cols;i++) {
        for (j=0;j<rows;j++) {
                *p++ = *nptr++;
        }
}
```

明示的にマッピングしたファイルを使用する場合の唯一の欠点は、書き込み時に生じます。 変更済みページを順次ブロックを使用して定期的に順番にファイルに書き込む、システムの遅延書き込みフィーチャーは、アプリケーションが **shmat()** または **mmap()** サブルーチンを使用した場合には、適用されません。 変更済みのページはメモリー内に集められ、仮想メモリー・マネージャー (VMM) がスペースを必要とする場合に、ランダムに書き込まれるだけです。 この状況によって、ディスクへの多くの少量の書き込みが生じる場合が多くなるので、CPU とディスクの使用が非効率的になります。

Java パフォーマンスのモニター

Java™ アプリケーションのボトルネックを切り分けて、パフォーマンスをチューニングするための幾つかの方法があります。

Java は、Oracle が開発したオブジェクト指向プログラミング言語です。この言語は C++ をモデルに設計され、プラットフォーム間およびオペレーティング・システム間でソース・レベルでもバイナリー・レベルでも移植できる、小さくて単純な言語になるように設計されています。アプレットおよびアプリケーションを含む Java プログラムは、Java 仮想マシン (JVM) がインストールされているマシン上で実行できます。

Java の利点

Java は、ほかの言語やほかの環境よりはるかに優れた点を備えており、それによってほとんどすべてのプログラミング・タスクに適するようになっています。

Java の優れた点は、次のとおりです。

- Java は学習するのが簡単です。

  Java は使いやすく設計されているので、作成、コンパイル、デバッグ、および学習が他のプログラム言語よりも簡単です。
- Java はオブジェクト指向です。

  これにより、使用者はモジュラー・プログラムと再使用可能コードを作成できます。
- Java はプラットフォームに依存しません。

  Java の最も優れている点の 1 つは、あるコンピューター・システムから別のコンピューター・システムへ容易に移動できることです。 同じプログラムを多数の異種システム上で実行できる能力を持つことは、 WWW のソフトウェアにとって極めて重要であり、Java は、ソースとバイナリーの両方のレベルでプラットフォームから独立することによって、それに成功しています。

Java の頑強性、使い易さ、クロスプラットフォーム機能、およびセキュリティー機能によって、世界中にインターネット・ソリューションを提供する、推奨する言語の 1 つになっています。

Java のパフォーマンスのガイドライン

AIX で Java パフォーマンスを改善する方法がいくつかあります。

- 最終的にはガーベッジ・コレクションを行う必要の生じるオブジェクトを不必要に作成するのを避けるために、過度の文字列操作を行う場合、文字列連結ではなく **StringBuffer** 機能を使用してください。
- Java コンソールへの書き込みが多くなり過ぎないようにして、文字列処理、テキストのフォーマット設定、および出力にかかるコストを減らしてください。
- 必要なときにプリミティブ型の変数を使用することによって、オブジェクトの生成と処理にあまりコストがかからないようにしてください。
- 使用頻度の高いオブジェクトをキャッシュに入れて、必要なガーベッジ・コレクションの量を減らし、オブジェクトを再作成しなくても済むようにしてください。
- 可能な場合には、ネイティブ・オペレーションをグループ化して、Java Native Interface (JNI) コールの数を減らしてください。
- 同期されたメソッドは必要なときにだけ使用して、 JVM およびオペレーティング・システムでのマルチタスキングを制限してください。
- 必要でない限り、ガーベッジ・コレクターを呼び出さないようにします。 呼び出す必要があるときには、アイドル時間中だけ、あるいはクリティカルでないフェーズでのみ、呼び出してください。
- 32 ビット・オペレーションは 64 ビット・オペレーションより速く実行されるので、可能な限り **long** 型ではなく **int** 型を使用してください。
- 可能な限り、メソッドは最終的なものとして宣言してください。 最終的なメソッドは JVM によって、より適切に処理されます。
- 変数を初期化する回数を減らすため、定数の作成時には、 **static final** キーワードを使用してください。
- Java でのキャストは実行時に行われるので、不要な「casts」と「instanceof」参照を使用しないでください。
- 配列で用が足りる場合にはいつでも、ベクトルは使用しないでください。
- ベクトルの端に項目を追加したり、そこから項目を削除したりしてください。
- ループ内でのオブジェクトの割り当ては避けてください。
- バッファー入出力を使用し、バッファー・サイズを調整してください。
- データベース・アクセスに対して接続プールと cached-prepared ステートメントを使用してください。
- データベースに対して接続プールを使用し、繰り返して接続をオープン/クローズしないで、接続を再利用してください。
- スレッドの作成および消滅のサイクルを最大化、最小化してください。
- 共用リソースの競合を最小化してください。
- ライフの短いオブジェクトの作成は最小にしてください。
- リモート・メソッド呼び出しは避けてください。
- ブロック化リモート・メソッド呼び出しを避けるためにコールバックを使用してください。
- メソッドのアクセスにのみ使用するオブジェクトの作成は避けてください。
- synchronized メソッドはループの外に出してください。
- 文字列および文字データはユニコードとしてデータベースに保管してください。
- CLASSPATH を並べ変えて、最も頻繁に使用されるライブラリーが前にくるように再配列してください。

Java モニター・ツール

ユーザーの Java アプリケーションには、パフォーマンス上悪影響を及ぼす要因のモニターおよび識別に使用できるツールがあります。

vmstat
いろいろなシステム・リソースについての情報を提供します。 これは、実行キューおよび待機キュー内のカーネル・スレッド、メモリー使用量、ページング・スペース、ディスク入出力、割り込み、システム・コール、コンテキスト・スイッチ、および CPU アクティビティーに関する統計情報を報告します。

iostat 詳細なディスク入出力情報を報告します。

topas CPU、ネットワーク、ディスク入出力、ワークロード・マネージャーおよびプロセス・アクティビティーを報告します。

tprof パフォーマンス上の問題を起こすと考えられるホット・ルーチンおよびメソッドの位置を正確に示すように、アプリケーションのプロファイルを作成します。

ps -mo *THREAD*
プロセスまたはスレッドがバインドされている CPU を示します。

Java profilers [-Xrunhprof, Xrunjpa64]
どのルーチンまたはメソッドが最も激しく使用されているかを判断します。

java -verbose:gc
アプリケーションにおけるガーベッジ・コレクションのインパクトをチェックします。 これは、ガーベッジ・コレクションに費やされた合計時間、ガーベッジ・コレクションあたりの平均時間、ガーベッジ・コレクションあたりに収集された平均メモリー、およびガーベッジ・コレクションあたりに収集された平均オブジェクトを報告します。

AIX のための Java のチューニング

AIX には、ユーザーの Java 環境のための推奨パラメーター・セットがあります。

AIXTHREAD_SCOPE=S
この変数のデフォルト値は **P** です。これはプロセス全体にわたるコンテンション有効範囲 (M:N) を意味します。値 **S** は、システム全体のコンテンション有効範囲 (1:1) を示します。 Java アプリケーションでは、この変数のデフォルト値は **S** です。

AIXTHREAD_MUTEX_DEBUG=OFF
デバッガーが使用するアクティブな mutex のリストを維持します。

AIXTHREAD_COND_DEBUG=OFF
デバッガーが使用する条件変数のリストを維持します。

AIXTHREAD_RWLOCK_DEBUG=OFF
デバッガーが使用するアクティブ相互排他ロック、条件変数、および読み取り/書き込みロックのリストを保存しています。 ロックが初期化されると、まだリストに入っていない場合は、リストに追加されます。 このリストはリンク・リストとしてインプリメントされているため、ロックが既にリストに入っているかどうかを判断するための検索は、リストが大きくなったときにパフォーマンスに影響を与えます。 この問題は、検索が行われている間、掛かったままになるロックでリストが保護されていることにより、さらに複雑になります。検索が行われている間、**pthread_mutex_init()** サブルーチンに対するその他の呼び出しは待たされます。 最適なパフォーマンスを得るには、このスレッド・デバッグ・オプションの値を、`OFF` に設定する必要があります。デフォルトは `ON` です。

SPINLOOPTIME=500

ブロッキングの前にプロセスがビジー・ロックに関してスピンできる回数です。 この値は、デフォルトでは 40 に設定されます。 **tprof** の出力が **check_lock** ルーチンでの高い CPU 使用率を示し、通常、短時間でのロックが使用可能な場合は、値を 500 以上に設定してスピン時間を増やす必要があります。

また、以下の設定を Java 環境用にお勧めします。

ulimit -d *unlimited*

ulimit -m *unlimited*

ulimit -n *unlimited*

ulimit -s *unlimited*

特定の環境パラメーターと設定値を使用すると、オペレーティング・システム内で Java のパフォーマンスをチューニングできます。 さらに、Java パフォーマンスを向上させるために、システム・コンポーネント (CPU、メモリー、ネットワーク、入出力など) をチューニングするための多くの手法が用意されています。 現在の状況に最も効果のある環境パラメーターを選択するための詳細については、個々のトピックを参照してください。

可能な限り最高の Java パフォーマンスとスケーラビリティーを得るには、オペレーティング・システムと Java のみならず、Just-In-Time (JIT) コンパイラーにも入手可能な最新バージョンを使用する必要があります。

Java のパフォーマンスに対するガーベッジ・コレクションの影響

Java に関連した最も一般的なパフォーマンス上の問題は、ガーベッジ・コレクション・メカニズムに関するものです。

Java ヒープのサイズが大きすぎる場合、そのヒープをメイン・メモリー外に置く必要があります。 ヒープのサイズが大きすぎると、ページング・アクティビティーの増加原因となり、Java のパフォーマンスに影響します。

また、大きなヒープを使用すると、そこにデータを充てんするのに数秒かかることがあります。 このことは、ガーベッジ・コレクションがそれほど頻繁に発生しなくても、ガーベッジ・コレクションによる休止時間が増加することを意味しています。

Java 仮想マシン (JVM) ヒープをチューニングするには、**-ms** または **-mx** オプション指定の **java** コマンドを使用します。 ガーベッジ・コレクションの統計情報を使用して、最適な設定値を判断してください。

トレース機能を使用したパフォーマンスの分析

オペレーティング・システムのトレース機能は、強力なシステム監視ツールです。

トレース機能は、タイム・スタンプ付きシステム・イベントの順次フローをキャプチャーし、システム・アクティビティーの詳細を細かいレベルで提供します。 イベントは、時間順、およびその他のイベントのコンテキスト順に示されます。 トレースは、システムとアプリケーションの実行を監視するための利用価値の高いツールです。 CPU 使用率や入出力待ち時間だけを提供するほかのツールとは違い、トレースは、その情報を展開して、どのようなイベントが発生しているのか、誰が責任者なのか、いつイベントが起こったのか、イベントがシステムにどのようにまたどのような理由で影響を与えるのかを理解する助けになります。

オペレーティング・システムは、システムの実行が全体として見やすくなるように計測機能を備えています。 ユーザーは、追加のイベントを挿入し、フォーマット設定規則を提供することによって、この見やすさをアプリケーションにも拡張することができます。

この機能の設計とインプリメンテーションには、トレース・データの収集が効果的に行われるように注意が払われたので、システムのパフォーマンスとフローは、トレースを活動化してもあまり変わりません。 このため、トレース機能は、パフォーマンス分析ツールとして、また問題判別ツールとして、非常に役立つものになっています。

トレース機能の詳細

トレース機能は、システムが維持する統計情報にアクセスして表示する従来のシステム・モニター・サービスより柔軟性に富んでいます。

トレース機能は、必要な統計情報を事前に予想する代わりに、イベントのストリームを提供するので、抽出する情報をユーザーが決定できます。従来のモニター・サービスでは、データ削減 (システム・イベントの統計情報への変換) の大部分は、システム・ツールと結び付いています。 例えば、多くのシステムでは、タスク A の実行に関して監視される最小、最大、および平均の経過時間が維持されるので、この情報を抽出することができます。

トレース機能は、データ削減をツールや手段に強く結合せずに、トレース・イベント・レコード (通常はイベント と省略します) のストリームを提供します。必要な統計情報を事前に決定する必要がありません。データ削減はツールなどとはほとんど切り離されています。 ユーザーは、タスク A に関する最小、最大、および平均時間を、イベントのフローから判別するよう選択できます。 しかし、次のようにすることも可能です。

- プロセス B によって呼び出された場合に、タスク A に関する平均時間を抽出する
- 条件 XYZ が真の場合に、タスク A に関する平均時間を抽出する
- タスク A の実行時間の標準偏差を計算する
- イベントのストリームによって認識された他のタスクの方が、要約する価値があると判断する

この柔軟性は、パフォーマンスまたは機能の問題を診断するために非常に重要です。

トレース機能を使用すれば、システム・アクティビティーに関する詳細情報を入手するだけでなく、アプリケーション・プログラムをツール化して、そのトレース・イベントをシステム・イベントに加えて収集することができます。 したがって、トレース・ファイルには、アプリケーションおよびシステムのアクティビティーの完全な記録が、正しい順序で、正確なタイム・スタンプと共に入ります。

トレース機能のインプリメンテーション

トレース・フック は、モニターする必要がある特定のイベントです。このイベントには、フック *ID* と呼ばれる固有の番号が割り当てられます。 **trace** コマンドは、これらのフックをモニターします。

trace コマンドは、ユーザー・プロセスおよびカーネル・サブシステムに関する統計情報を生成します。バイナリー情報は、メモリー内の 2 つの代替バッファーに書き込まれます。 次に、**trace** プロセスは、情報をディスク上のトレース・ログ・ファイルに転送します。 このファイルは急速に大きくなります。**trace** プログラムは、**ps** コマンドによってモニター可能なプロセスとして実行します。 **trace** コマンドは、アカウンティングと同様に、デーモンとして機能します。

次の図は、トレース機能のインプリメンテーションを示しています。

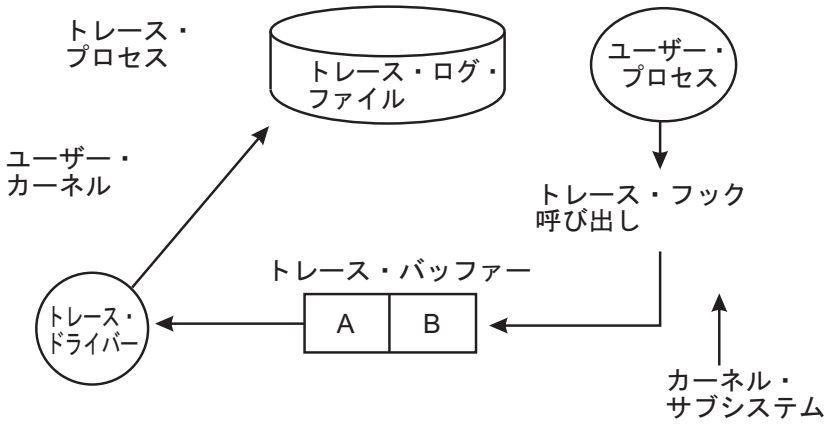


図 25. トレース機能のインプリメンテーション： この図は、トレースのプロセスを示しています。 このプロセスで、ユーザー・プロセス (カーネル・サブシステム) はトレース・フック呼び出しをラベル A および B のトレース・バッファーに送信します。呼び出しはバッファーからトレース・ドライバーを経由して、ユーザー・カーネルのトレース・ログ・ファイルに記録されます。

モニター機能はシステム・リソースを使用します。 理想を言えば、オーバーヘッドは、システムの実行に重大な影響を与えない程度に小さくなければなりません。 **trace** プログラムがアクティブである場合の CPU のオーバーヘッドは、 2% より小さいものです。 トレース・データでバッファーが満杯になり、ログに書き込む必要が生じた場合には、ファイル入出力のために追加の CPU が必要です。通常、これは 5% より小さいものです。 **trace** プログラムはバッファー・スペースを要求し占有するので、メモリーに制約がある環境では、かなり大きなパーセントになることがあります。 トレース・ログおよびレポート・ファイルが非常に大きくなる可能性があることに注意してください。

収集するトレース・データの量の制限

トレース機能は、大量のデータを生成します。 このデータを長時間取り込むと、ストレージ・デバイスがオーバーフローします。

トレース機能を効率的に使用するには、以下の 2 つの方法があります。

- 複数の方法でトレース機能をオンまたはオフにして、システム・アクティビティーを取り込むことができます。 この方法で、数秒から数分のシステム・アクティビティーを取り込んで、後処理することは実用的です。 このくらいの時間で、主要なアプリケーション・トランザクションまたは長いタスクの重要なセクションの特徴を記述するのに十分です。
- トレース機能は、イベント・ストリームを標準出力に送信するように構成することができます。 これによって、リアルタイム・プロセスをイベント・ストリームに接続し、イベントの記録時にデータ削減を行って、長時間のモニター能力を生み出すことができます。 特化したツールの論理的な拡張は、大量のデータの保管または動的なデータ削減が可能な補助デバイスへデータ・ストリームを振り向けることです。 この手法は、パフォーマンス・ツール **tprof**、**pprof**、**netpmon**、および **filemon** によって使用されます。

トレースの開始と制御

トレース機能は、以下の 3 つの異なる使用モードを提供します。

サブコマンド・モード

トレースは、シェル・コマンド (**trace**) で開始し、サブコマンドを介してユーザーとのダイアログを続行します。 元のシェル・プロセスが使用中なので、トレース対象のワークロードは、他のプロセスが提供する必要があります。

コマンド・モード
: トレースは、トレース機能の非同期的な実行を指定するフラグを含んでいるシェル・コマンド (**trace -a**) で開始します。元のシェル・プロセスは、トレース制御コマンドが中に散在して入っている通常のコマンド群を自由に実行することができます。

アプリケーション制御モード
: トレースは **trcstart()** サブルーチンで開始し、アプリケーション・プログラムからの **trcon()** および **trcoff()** などのサブルーチン呼び出しによって制御します。

トレース・データのフォーマット設定

汎用トレース報告機能は、**trcrpt** コマンドによって提供されます。

報告機能は、データ削減はほとんど行いませんが、ロー・バイナリー・イベント・ストリームを読み取り可能な ASCII リストに変換します。 データはリーダーによって表示可能なフォーマットで抽出することができ、あるいはデータをさらに減らすためのツールを作成することができます。

報告機能は、トレース・フォーマット・ファイルに指定された規則に従って、各イベントのテキストとデータを表示します。 デフォルトのトレース・フォーマット・ファイルは `/etc/trcfmt` であり、これには各イベント ID のグループが含まれています。 各イベントのグループは、報告機能に、そのイベントのフォーマット設定規則を提供します。 この手法によって、ユーザーは固有のイベントをプログラムに追加し、対応するイベント・グループをフォーマット・ファイル内に挿入して、新規イベントのフォーマット設定方法を指定することができます。

トレース・データの表示

トレース・データをフォーマット設定すると、特定のイベントに関するすべてのデータが通常は 1 行に置かれます。

追加の行に、説明情報が含まれている場合もあります。 組み込まれているフィールドによっては、フォーマット済みの行が 80 文字を超える可能性もあります。レポートは、132 桁をサポートする出力デバイスに表示することをお勧めします。

トレース機能の使用例

標準のトレースには、トレース・ファイルの取得、フォーマット設定、フィルター処理、および読み取りが含まれます。

注: この例は、入力ファイルがシステム・メモリー内のキャッシュにまだ入っていない場合に、特に意味があります。 ソース・ファイルとして、サイズが約 50 KB で、最近使用されていないファイルを選択してください。

サンプル・トレース・ファイルの取得

トレース・データは急速に累積されます。 データ収集を、関心のある領域のできるだけ近くに限定してください。 それを行うための 1 つの手法は、同じコマンド・ラインで複数のコマンドを発行することです。

次に例を示します。

```
# trace -a -k "20e,20f" -o trc_raw ; cp ../bin/track /tmp/junk ; trcstop
```

cp コマンドの実行をキャプチャーします。 ここでは、**trace** コマンドの 2 つのフィーチャーを使用しています。 **-k "20e,20f"** オプションは、**lockl()** および **unlockl()** 関数からのイベントの収集を抑制します。これらの呼び出しは、ユニプロセッサー・システムでは非常に数が多くなりますが、 SMP システムでは

少なくなります。追加情報なしでレポートにボリュームを追加します。 **-o trc_raw** オプションは、ロー・トレース出力ファイルをローカル・ディレクトリーに書き込むようにします。

サンプル・トレースのフォーマット設定

トレース・レポートをフォーマット設定する場合は、**trcrpt** コマンドを使用します。

```
# trcrpt -O "exec=on,pid=on" trc_raw > cp.rpt
```

これは、実行されるファイルの完全修飾名と、それに割り当てられたプロセス ID の両方を報告します。

レポート・ファイルは、トレース内に非常に多くの VMM ページの割り当ておよび削除のイベントがあることを、以下のように示します。

```
1B1 ksh            8526          0.003109888       0.162816
         VMM page delete:      V.S=0000.150E ppage=1F7F
             working_storage delete_in_progress process_private computational

1B0 ksh            8526          0.003141376       0.031488
         VMM page assign:      V.S=0000.2F33 ppage=1F7F
             working_storage delete_in_progress process_private computational
```

現時点では、この詳細レベルの VMM アクティビティーには関心がないので、トレースを以下のように再フォーマット設定します。

```
# trcrpt -k "1b0,1b1" -O "exec=on,pid=on" trc_raw > cp.rpt2
```

-k "1b0,1b1" オプションは、フォーマット設定済み出力では不要な VMM イベントを抑制します。これによって、不要なイベントを抑制するために、ワークロードを再トレースする必要がなくなります。ロック・アクティビティーを特定の時点で調べる必要が生じると予想される場合には、**trace** コマンドの **-k** 機能を使用する代わりに、**trcrpt** コマンドの同じ機能を使用して、**lockl()** および **unlockl()** イベントを抑制することができます。 イベントの小さなセットにのみ関心がある場合には、 **-d"***hookid1,hookid2***"** を指定すれば、指定した範囲内のイベントのみに関するレポートを生成することができます。 フック ID はレポートの左端の列なので、組み込みまたは除外の対象であるフックのリストを迅速にコンパイルできます。トレース・フック ID の包括的なリストは、/usr/include/sys/trchkid.h ファイルに定義します。

トレース・レポートの読み取り

トレース・レポートのヘッダーは、トレースが実行された時期と場所、およびトレースの生成に使用されたコマンドを示します。

以下はサンプルのヘッダーです。

```
Thu Oct 28 13:34:05 1999
System: AIX texmex Node: 4
Machine: 000691854C00
Internet Protocol Address: 09359BBB 9.53.155.187
Buffering: Kernel Heap

trace -a -k 20e,20f -o trc_raw
```

レポートのボディは、十分に小さいフォントで表示した場合には、以下のようになります。

```
ID  PROCESS NAME  PID     ELAPSED_SEC    DELTA_MSEC  APPL  SYSCALL KERNEL  INTERRUPT
101 ksh           8526    0.005833472    0.107008          kfork LR = D0040AF8
101 ksh           7214    0.012820224    0.031744          execve LR = 10015390
134 cp            7214    0.014451456    0.030464          exec:  cmd=cp ../bin/track /tmp/junk pid=7214 tid=24713
```

cp.rpt2 で、以下の情報を見ることができます。

- **cp** プロセスの **fork()**、**exec()**、およびページ・フォールトのアクティビティー。

- 読み取りのための入力ファイルのオープンおよび /tmp/junk ファイルの作成。
- コピーを実行するための、連続する **read()**/**write()** システム・コール。
- 入出力完了を待っている間にブロック状態になったプロセス **cp**、およびディスパッチ中の **wait** プロセス。
- 論理ボリューム要求を物理ボリューム要求に変換する方法。
- ファイルが従来のカーネル・バッファーに入れられずに、マップされ、読み取りアクセスによって、仮想メモリー・マネージャーが解決しなければならないページ・フォールトが引き起こされたこと。
- 仮想メモリー・マネージャーが、順次アクセスをセンスし、ファイル・ページの事前取り出しを開始したこと。
- 順次アクセスの継続に従って、事前取り出しのサイズが大きくなったこと。
- 可能な場合、ディスク装置ドライバーが、ドライブに対する複数のファイル要求を 1 つの入出力要求に合体したこと。

最初、トレース出力は多すぎると感じます。 これは、学習の補助として使用するために適している例です。 上記のアクティビティーを識別できれば、トレース機能を使用してシステム・パフォーマンス問題を診断できるようになります。

トレース・レポートのフィルター操作

完全に詳細なトレース・データが必要でない場合があります。 表示が必要な、関心のある特定のイベントを選択することができます。

例えば、特定のイベントの発生回数を調べることが役に立つ場合があります。 「コピー例内で発生したオープンの回数は ?」という質問に答えるには、最初に、**open()** システム・コールのイベント ID を調べます。 これは、以下のように行うことができます。

```
# trcrpt -j | grep -i open
```

イベント ID 15b が OPEN SYSTEM CALL イベントであることが分かるはずです。 次に、以下のようにしてコピー例からのデータを処理してください。

```
# trcrpt -d 15b -O "exec=on" trc_raw
```

レポートが標準出力に書き出されるので、発生した **open()** サブルーチンの数を判別することができます。 **cp** プロセスが実行した **open()** サブルーチンのみを表示したい場合には、以下のコマンドを使用して、報告コマンドを再び実行してください。

```
# trcrpt -d 15b -p cp -O "exec=on" trc_raw
```

コマンド・ラインからのトレースの開始と制御

trace コマンドによって、トレース機能を構成し、データ収集をオプションで開始することができます。このコマンドの詳細な構文の説明は*Commands Reference, Volume 5*にあります。

trace コマンドによってトレースを構成した後は、データ収集をオン/オフにしたりトレース機能を停止するための制御があります。このような制御を起動するには、サブコマンド、コマンド、およびサブルーチンを使用します。サブルーチン・インターフェースについては、 432 ページの『プログラムからのトレースの開始と制御』を参照してください。

サブコマンド・モードにおけるトレース制御

-a オプションを指定しないで **trace** ルーチンを構成すると、このルーチンはサブコマンド・モードで実行されます。

サブコマンド・モードでトレース・ルーチンを実行する場合は、通常のシェル・プロンプトの代わりに "->" のプロンプトが表示されます。 このモードでは、以下のサブコマンドが認識されます。

trcon イベント・データの収集の開始または再開

trcoff イベント・データの収集の延期

q または **quit**
イベント・データの収集の停止および **trace** ルーチンの終了

!command
指定されたシェル・コマンドの実行

? 使用可能コマンドの表示

次に例を示します。

```
# trace -f -m "Trace of events during mycmd"
-> !mycmd
-> q
#
```

コマンドによるトレース制御

trace ルーチンの制御に使用できる幾つかのコマンドがあります。

trace ルーチンを非同期的に実行するように構成した場合 (**trace -a**)、トレースを以下のコマンドによって制御することができます。

trcon イベント・データの収集の開始または再開

trcoff イベント・データの収集の延期

trcstop
イベント・データの収集の停止および **trace** ルーチンの終了

次に例を示します。

```
# trace -a -n -L 2000000 -T 1000000 -d -o trace.out
# trcon
# cp /a20kfile /b
# trcstop
```

-d (**trcon** サブコマンドの入力までのトレースの据え置き) オプションを指定すれば、**trace** コマンド自体で実行するトレースの量を制限することができます。 **-d** オプションを指定しない場合、トレースは即時に開始され、固有のメモリー・バッファーを初期化中の **trace** コマンドに関するイベントを記録することができます。 一般的には、**trace** コマンド自身を除くすべてのものをトレースする必要があります。

デフォルトで、カーネル・バッファー・サイズ (**-T** オプション) は、ログ・バッファー・サイズ (**-L** オプション) の半分までにすることができます。 **-f** フラグを使用すれば、両方のバッファー・サイズを同じにすることができます。

-n オプションは、トレースする必要があるカーネル・エクステンション・システム・コールがある場合に役立ちます。

プログラムからのトレースの開始と制御

トレース機能は、サブルーチン呼び出しを介して、プログラムから開始することができます。 このサブルーチンは **trcstart()** であり、 `librts.a` ライブラリーに入っています。

trcstart() サブルーチンの構文を以下に示します。

```
int trcstart(char *args)
```

ここで *args* は、**trace** コマンド用に入力してあるオプション・リストです。デフォルトで、システム・トレース (channel 0) が開始されます。 汎用トレースを開始したい場合には、 *args* 文字列に **-g** オプションを組み込んでください。 正常終了した時点で、**trcstart()** サブルーチンはチャネル ID を戻します。 汎用トレースの場合、このチャネル ID を使用すれば、専用の汎用チャネルに記録することができます。

このサブルーチンを使用してプログラムをコンパイルする場合には、 `librts.a` ライブラリーへのリンクを特に要求する必要があります (コンパイル・オプションとして **-l rts** を使用してください)。

トレース・サブルーチン呼び出しによるトレース制御

trace ルーチンの制御は、 `librts.a` ライブラリーからのサブルーチンとして使用することができます。

これらのサブルーチンは、正常終了するとゼロを戻します。 以下のサブルーチンがあります。

int trcon()
: トレース・データの収集の開始または再開。

int trcoff()
: トレース・データの収集の延期。

int trcstop()
: トレース・データの収集の停止および **trace** ルーチンの終了。

trcrpt コマンドを使用したレポートのフォーマット設定

トレース報告機能は、トレース・ログ・ファイルを読み取り、トレース・エントリーをフォーマット設定し、レポートを作成します。

trcrpt コマンドは、トレース・フォーマット・ファイル (`/etc/trcfmt`) に指定された規則に従って、各イベントのテキストとデータを表示します。フォーマット・ファイル内のグループは、イベントまたはフックに関するフォーマット設定規則を示します。 プログラムにフックを追加するユーザーは、対応するイベント・グループをフォーマット・ファイルに挿入して、イベントのトレース・データを印刷することができます (435 ページの『トレース・イベントの新規追加』を参照してください)。

trcrpt 機能は要約レポートを作成しませんが、**awk** コマンドを使用すれば、**trcrpt** 出力をさらに処理することによって、簡単な要約を作成することができます。

trcrpt コマンドの詳細な構文の説明は、 *Commands Reference, Volume 5*にあります。

同じシステム上のレポートのフォーマット設定

trcrpt コマンドは、トレース・ログ・ファイルに入っているトレース・イベント・データのレポートのフォーマットを設定します。

このコマンドを使用すれば、レポートに組み込む (または組み込まない) イベントを指定したり、出力のプレゼンテーションを指定したりすることができます。

システム管理インターフェース・ツール (SMIT) を使用すれば、SMIT 高速パスを入力することによって、**trcrpt** コマンドを実行することができます。

```
# smitty trcrpt
```

トレース・レポートを `newfile` ファイルに作成するには、以下のように入力してください。

```
# trcrpt -o newfile
```

さまざまなシステム上でのレポートのフォーマット設定

trcrpt コマンドは、トレースを収集したシステムとは別のシステムで実行する方が望ましい場合があります。

これには、以下のようなさまざまな理由があります。

- トレース対象のシステムが、**trcrpt** コマンドを実行するために使用できない場合があります。また、トレースがリモート・サイトのシステム管理者などによって収集される場合があります。
- トレース対象のシステムがビジーであるために、**trcrpt** コマンドを実行できない場合があります。
- トレース対象のシステムに、非常に大きな **trcrpt** ファイルを収容するための十分なファイルシステム・スペースが残っていない場合があります。

あるシステムで **trace** コマンドを実行し、そのトレース・ファイルに対して別のシステムで **trcrpt** コマンドを実行することができます。 これが正しく機能するためには、**trcnm** コマンドの出力が、トレースを実行したシステムから入手できなければなりません。 **trcnm** コマンドを実行し、出力を以下のようにファイルにリダイレクトしてください。

```
# trcnm > trace.nm
```

tprof、**pprof**、**netpmon**、および **filemon** などの他のパフォーマンス・ツールでトレース・ファイルを使用する必要がある場合には、**gennames** *Gennames_File* コマンドを実行してください。

そのファイルは、次に **trcrpt** コマンドの **-n** フラグと共に、以下のように使用します。

```
# trcrpt -n trace.nm -o newfile
```

-n を指定しない場合、**trcrpt** コマンドは、**trcrpt** コマンドを実行するシステムから記号テーブルを生成します。

さらに、トレース対象のシステムが、**trcrpt** コマンドを実行するシステムと異なるトレース・フォーマット・グループ、または複数のトレース・フォーマット・グループを持っている場合があるので、トレース対象のシステムからの `/etc/trcfmt` ファイルのコピーは有用であることがあります。 **trcrpt** コマンドで **-t** フラグを使用すれば、トレース・フォーマット・ファイルを指定することができます (デフォルトでは、**trcrpt** コマンドは、このコマンドを実行中のシステムからの `/etc/trcfmt` ファイルを使用します)。次に例を示します。

```
# trcrpt -n trace.nm -t trcfmt_file -o newfile
```

トレース **-C** 出力によるレポートのフォーマット設定

-C フラグを指定してトレースを実行した場合、1 つ以上のトレース出力ファイルが生成されます。

例えば、トレース・ファイル名を `trace.out` と指定し、4-way SMP で **-C all** を指定した場合、`trace.out`、 `trace.out-1`、`trace.out-2`、`trace.out-3`、および `trace.out-4` の各ファイルが生成されます。 **trcrpt** コマンドを実行する場合には、以下のように **trcrpt -C all** と指定し、ファイル名として `trace.out` を指定すれば、すべてのファイルが読み取られます。

```
# trcrpt -C all -r trace.out > trace.tr
```

次に、この `trace.tr` ファイルを、他のコマンドの入力として使用することができます (このファイルは各 CPU からのトレース・データが組み込まれています)。 トレースで **-C** フラグを指定する理由は、多くの (例えば、12 を超える) CPU を持っているシステム上で、トレースが各 CPU のアクティビティーに遅れ

ないようにするためです。 もう 1 つの理由は、**-C all** フラグを指定した場合、トレース・バッファーのバッファー・サイズが各 CPU ごとになるからです。

トレース・イベントの新規追加

オペレーティング・システムは、重要なイベントが組み込まれた状態で出荷されます。 オペレーティング・システムからイベントのフローを取り込むために、ユーザーはトレースを活動化するだけで済みます。アプリケーション開発者は、開発中に、チューニングのために、アプリケーション・コードを組み込むことができます。 これによって、アプリケーション開発者は、アプリケーションがシステムと対話する方法を知ることができます。

トレース・イベントを追加するには、プログラムが生成するトレース・レコードを、トレース・インターフェースの規則に従って設計する必要があります。 次に、トレース・フック・マクロをプログラムの適切な位置に追加します。 これにより、トレースを起動し制御するための標準の方法 (コマンド、サブコマンド、またはサブルーチン呼び出し) の内のいずれかを使用して、トレースを実行することができます。**trcrpt** プログラムを使用してトレースをフォーマット設定するには、各新規トレース・レコードおよびそのフォーマット設定要件について記述するグループを、トレース・フォーマット・ファイルに追加してください。

トレース・イベント・レコードの可能な形式

イベントは、フック・ワード、オプショナルのデータ・ワード、およびタイム・スタンプからなります。

下図に示すように、イベント・レコードが使用できる形式ごとに、4 ビットのタイプが定義されます。 タイプ・フィールドは記録ルーチンによって記入されるので、トレース・フォーマット・ファイル内のフォーマット設定規則が正しくない場合、または当該イベントに関するフォーマット設定規則がない場合でも、報告機能はデータの処理時にどのイベント間でも常にスキップすることができます。

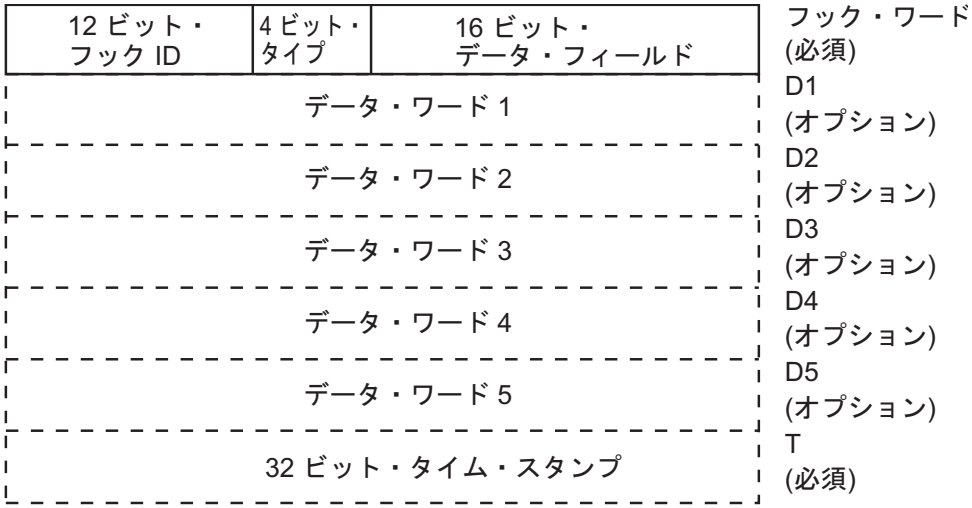


図 *26.* トレース・イベント・レコードのフォーマット： この図は、7 つの行がある表です。 1 行目のセルは、「12 ビットのフック ID」、「4 ビットのタイプ」、「16 ビットのデータ・フィールド」です。 次の 6 行は「データ・ワード 1」から「データ・ワード 5」で、最後の行は「32 ビットのタイム・スタンプ」です。 行 1 の行見出しは、「フック・ワード (必須)」です。次の 5 行の見出しは、「D1 (オプション)」、「D2 (オプション)」、「D3 (オプション)」、「D4 (オプション)」、および「D5 (オプション)」です。 最後の行ラベルは「T (必須)」です。

イベント・レコードはできるだけ短くする必要があります。 多くのシステム・イベントはフック・ワードとタイム・スタンプしか使用しません。 長形式によって、ユーザーは可変長のデータを記録することができます。 この長形式では、フック・ワードの 16 ビットのデータ・フィールドが、イベント・レコードの長さを記述する長さフィールドに変換されます。

トレース・チャネル

トレース機能では、トレース・フックのアクティビティーに関する最高 8 つまでのチャネル (0 から 7 の番号が付きます) を同時に提供することができます。

チャネル 0 は常にシステム・イベントに関して使用されますが、アプリケーション・イベントがこのチャネルを使用することもできます。 汎用チャネルと呼ばれる他の 7 つのチャネルは、アプリケーション・プログラムのアクティビティーをトレースするために使用することができます。

トレースを開始すると、デフォルトでチャネル 0 が使用されます。 **trace -n** *channel_number* コマンドは、汎用チャネルへのトレースを開始します。 汎用チャネルの使用には、いくつかの制限があります。

- 汎用チャネルへのインターフェースの場合、チャネル間の区別が必要であり、汎用チャネルは可変長レコードを記録するので、チャネル 0 へのインターフェースより多くの CPU 時間が必要です。
- チャネル 0 および汎用チャネルに記録されたイベントは、シーケンスによってではなく、タイム・スタンプによってのみ相関関係が付けられるので、最初にどのイベントが発生したのかを判別できない場合が生じる可能性があります。

トレース・イベントを記録するためのマクロ

イベント・レコードの可能な各タイプを記録するためのマクロは、/usr/include/sys/trcmacros.h ファイルに定義されています。

イベント ID は、/usr/include/sys/trchkid.h ファイルに定義されています。これらの 2 つのファイルを、トレース・イベントを記録するすべてのプログラムに組み込んでください。

タイム・スタンプと共にイベントをチャネル 0 に記録するには、以下のマクロを使用します。

```
TRCHKL0T(hw)
TRCHKL1T(hw,D1)
TRCHKL2T(hw,D1,D2)
TRCHKL3T(hw,D1,D2,D3)
TRCHKL4T(hw,D1,D2,D3,D4)
TRCHKL5T(hw,D1,D2,D3,D4,D5)
```

AIX の旧バージョンでは、チャネル 0 のイベントをタイム・スタンプなしで記録するには、以下のマクロを使用します。

```
TRCHKL0(hw)
TRCHKL1(hw,D1)
TRCHKL2(hw,D1,D2)
TRCHKL3(hw,D1,D2,D3)
TRCHKL4(hw,D1,D2,D3,D4)
TRCHKL5(hw,D1,D2,D3,D4,D5)
```

すべてのトレースはマクロの使用に関係なくタイム・スタンプが記録されます。

トレース・イベント・レコードのタイプ・フィールドは、hw パラメーター内の 4 ビットの値に関係なく、使用したマクロに対応する値に設定されます。

汎用チャネル (1 から 7) のいずれかにイベントを記録するマクロは 2 つだけです。 これらのマクロを以下に示します。

```
TRCGEN(ch,hw,D1,len,buf)
TRCGENT(ch,hw,D1,len,buf)
```

これらのマクロは、チャネル・パラメーター (ch) によって指定されたイベント・ストリームに、フック・ワード (hw)、データ・ワード (D1)、およびユーザーのデータ・セグメントからの buf によって指定された位置から開始する len バイトを記録します。

イベント ID の使用

トレース・レコード内のイベント ID は、そのレコードを、レコードの特定のクラスに属しているものとして識別します。 イベント ID は、トレース・メカニズムがトレース・フックの記録または無視を決定する基礎であり、フォーマット設定済みのレポートへのトレース・レコードの組み込みまたは除外を **trcrpt** コマンドが決定する基礎でもあります。

AIX 6.1 より前、および AIX 6.1 以降で稼働している 32 ビット・アプリケーションでは、考えられる 4096 の ID のイベント ID は 12 ビット (3 桁の 16 進数字) です。 予約済みで、コードと共に出荷されるイベント ID は、重複を回避するために永続的に割り当てられています。 ユーザーが自分の環境内に、あるいは開発中にイベントを定義できるように、0x010 から 16 進数 0x0FF の範囲のイベント ID が一時使用のために予約されています。 ユーザーはこの範囲内の ID を独自の環境 (すなわち、同一のイベント ID があいまいに使用されないことをユーザーが保証する準備ができているシステムのセット) 内で自由に使用することができます。

AIX 6.1 以降を実行している 64 ビット・アプリケーションおよびカーネル・ルーチンでは、考えられる 65536 の ID に対して 16 ビットのイベント ID (4 桁の 16 進数字) を使用することができます。 0x1000 未満のイベント ID の最下位数字は 0 でなければなりません ("0x0hh0" の形式)。 ユーザーが自分の環境内に、あるいは開発中にイベントを定義できるように、0x0100 から 0x0FF0 の範囲のイベント ID は一時使用のために予約されています。

注: このイベントの範囲を使用する各ユーザーが、コードがユーザーの環境から出ないようにすることが重要です。 ID の使用を制御できない環境に、一時フック ID で組み込んだコードを配布する場合には、自分の環境で同じ ID を既に使用している他のプログラムと衝突するリスクが生じることになります。

イベント ID は、数が少ないので保存する必要があります。しかし、16 ビットのデータ・フィールドを使用することによって拡張することができます。 これによって、公式のフック ID ごとに 65536 個の区別可能なイベントを生成できます。 固有の ID を持つ唯一の理由は、ID が、トレース機能内で収集および報告のフィルター操作が使用可能なレベルを示すからです。

ユーザーが追加したイベントは、指定したトレース・フォーマット・ファイル内にそのイベントに関するグループがあれば、 **trcrpt** コマンドによってフォーマット設定することができます。 トレース・フォーマット・ファイルは、編集可能な ASCII ファイルです (439 ページの『トレース・フォーマット・ファイル内のグループの構文』を参照してください)。

イベントのコーディングおよびフォーマット設定の例

プログラム・ループの実行時間を測定するためのトレース・イベントの使用法です。

```
#include <sys/trcctl.h>
#include <sys/trcmacros.h>
#include <sys/trchkid.h>
char *ctl_file = "/dev/systrctl";
int ctlfd;
int i;
main()
{
  printf("configuring trace collection ¥n");
  if (trcstart("-ad")){
```

```
    perror("trcstart");
    exit(1);
  }

  printf("opening the trace device  ¥n");
  if((ctlfd = open(ctl_file,0))<0){
    perror(ctl_file);
    exit(1);
  }

  printf("turning  trace on ¥n");
  if(ioctl(ctlfd,TRCON,0)){
    perror("TRCON");
    exit(1);
  }

  for(i=1;i<11;i++){
    TRCHKL1T(HKWD_USER1,i);

    /* The code being measured goes here. The interval */
    /* between occurrences of HKWD_USER1 in the trace  */
    /* file is the total time for one iteration.       */
  }

  printf("turning trace off¥n");
  if(ioctl(ctlfd,TRCSTOP,0)){
    perror("TRCOFF");
    exit(1);
  }

  printf("stopping the trace daemon ¥n");
  if (trcstop(0)){
    perror("trcstop");
    exit(1);
  }

  exit(0);
}
```

サンプル・プログラムをコンパイルする場合、以下のように librts.a ライブラリーにリンクする必要があります。

```
# xlc -O3 sample.c -o sample -l rts
```

HKWD_USER1 はイベント ID で、値は 16 進数で 010 です (/usr/include/sys/trchkid.h ファイルを調べれば確認できます)。 報告機能は、トレース・フォーマット・ファイルに規則が示されていない場合には、HKWD_USER1 イベントをフォーマット設定しません。 以下の HKWD_USER1 に関するグループの例を使用することができます。

```
# User event HKWD_USER1 Formatting Rules Stanza
# An example that will format the event usage of the sample program
010 1.0 L=APPL "USER EVENT - HKWD_USER1" O2.0      ¥n ¥
               "The # of loop iterations =" U4     ¥n ¥
               "The elapsed time of the last loop = " ¥
                endtimer(0x010,0x010) starttimer(0x010,0x010)
```

グループの例を入力する場合、マスター・フォーマット・ファイル /etc/trcfmt を変更しないでください。その代わりに、コピーを作成して、ユーザー所有のディレクトリー (例えば、**mytrcfmt** という名前を付けます) に保持してください。 サンプル・プログラムを実行した場合、**trcstart()** サブルーチンに対して他のログ・ファイルを指定していないので、ロー・イベント・データがデフォルトのログ・ファイルに取り込まれます。 出力レポートをフィルター処理すれば、ユーザーのイベントのみを入手することができます。 これを行うには、以下のように **trcrpt** コマンドを実行してください。

```
# trcrpt -d 010 -t mytrcfmt -O "exec=on" > sample.rpt
```

sample.rpt ファイルをブラウズすれば、結果を表示することができます。

トレース・フォーマット・ファイル内のグループの構文

トレース・フォーマット・ファイルは、各イベント ID に関して予期されるデータのプレゼンテーションおよび表示の規則を提供します。 これによって、報告機能を変更せずに、新規イベントをフォーマット設定できます。

新規イベントの規則は単純にフォーマット・ファイルに追加されます。 規則の構文によって、データのプレゼンテーションに柔軟性を与えることができます。

トレース・フォーマットのグループは、特定のイベントの規則を記述するために必要なだけ長くすることができます。 グループは、現在の行を '¥' 文字で終了することによって、次の行に継続することができます。 各フィールドの説明はファイル参照にあります。

/etc/trcfmt ファイル内のコメントは、他のフォーマットとマクロの可能性について示し、ユーザーが追加のマクロを定義する方法を説明しています。

レポート作成パフォーマンス上の問題

オペレーティング・システムにパフォーマンス上の問題の可能性を検出した場合には、問題を報告し、問題分析データを提供するためのツールおよび手順を使用することができます。 これらのツールの目的は、ユーザーが最小の労力と時間で、プロンプトおよび正確な応答を入手することです。

ベースラインの測定

パフォーマンス上の問題は、しばしばシステム・ハードウェアまたはソフトウェアに対するなんらかの変更の直後に報告されます。 変更後のパフォーマンスと比較するための、変更前のベースライン測定がなければ、問題の定量化は不可能です。

以下のいずれかに対する変更はパフォーマンスに影響を与える可能性があります。

- ハードウェア構成 - ディスクの接続方法のような構成の追加、除去、または変更
- オペレーティング・システム - ファイルセットのインストールまたは更新、 PTF のインストール、およびパラメーターの変更
- アプリケーション - 新規バージョンおよび修正のインストール
- アプリケーション - データ配置の構成または変更
- アプリケーションのチューニング
- オペレーティング・システム、RDBMS またはアプリケーション内のオプションのチューニング
- すべての変更

最良の選択は、変更の前後に環境の測定を行うことです。 別の方法では、一定の間隔で (例えば、1 月に 1 回) 測定を行い、出力を保管します。 問題が検出された場合、以前の測定結果を使用して比較することができます。 一連の出力を収集することは、発生する可能性があるパフォーマンス上の問題の診断をサポートするために有用です。

パフォーマンス診断の効果を最大化するには、パフォーマンスが問題になる可能性がある作業日、週、または月のさまざまな期間に関して、データを収集してください。 例えば、以下のようなワークロードのピークが存在する可能性があります。

- オンライン・ユーザーに関しては、午前中の真ん中
- 深夜のバッチ実行時

- 月末処理時
- 主要なデータのロード時

測定機能を使用して上記の各ワークロードのピーク時ごとにデータを収集してください。その理由は、パフォーマンス上の問題が、他の時間ではなく、上記期間のいずれかの間でのみ問題の原因となっている可能性があるからです。

注**:** すべての測定は、測定対象のシステムのパフォーマンスに影響を与えます。
AIX パフォーマンス PMR (perfpmr) データ収集ツールが、ベースラインとなるデータの収集には好ましい方式と言えます。 これらのツールは、web アドレスの ftp://ftp.software.ibm.com/aix/tools/perftools/perfpmr でアクセスできます。測定対象の AIX バージョンと一致するディレクトリーにある README ファイルの指示に従って、ツールの入手、インストール、およびシステムでのデータ収集を行ってください。

パフォーマンス上の問題とは何か

サポート担当者は、報告された問題が機能上の問題であるか、パフォーマンス上の問題であるかを判別する必要があります。

アプリケーション、ハードウェア・システム、またはネットワークが正しく動作していない場合、これは、機能上の問題 と呼ばれます。 例えば、メモリー・リークが発生しているアプリケーションまたはシステムには機能上の問題があります。

機能上の問題は、パフォーマンス上の問題の原因になる場合があります。例えば、機能が実行されているが、機能の速度が遅い場合です。 このような場合には、システムをチューニングするよりも、問題の根本的な原因を判別して、それを修正することが重要です。 別の例は、ダウンしたネットワークまたはネームサーバーのために、通信速度が遅くなった場合です。

パフォーマンス上の問題の記述

サポート担当員は、誰かがシステムに関するパフォーマンス上の問題を検出したことを示し、なんらかのデータ分析を提供している問題レポートを受け取る場合があります。この情報は、パフォーマンス上の問題の性質を正確に判別するには不十分です。 データが 100% の CPU 使用率および高い実行キューを示している場合でも、パフォーマンス上の問題の原因とは無関係である可能性があります。

例えば、システムに、ネットワーク上のリモート端末から幾つかのルーターを経てログインしているユーザーがいるとします。 このユーザーが、システムが遅いと報告するとします。 データは、CPU が非常に激しく使用されていることを示す場合があります。 しかし、実際の問題は、ネットワーク上でパケットが脱落したために (この原因は、ルーターの失敗またはネットワークの過負荷である可能性があります)、端末上での長い遅延の後に文字が表示された可能性があります。 この状態は、マシンの CPU 使用率とは無関係であることがあります。 一方、苦情が、システム上でのバッチ・ジョブの実行に時間がかかったということであれば、CPU の使用率または入出力の処理能力が関係している可能性があります。

データの収集または分析を試みる前に、パフォーマンス上の問題に関する以下の質問に答えることによって、できるだけ多くの詳細を常に入手してください。

- 特定のコマンドを実行するか、またはイベントのシーケンスを再構成することによって、問題を再現できますか。 (例えば、**ls** `/slow/fs` または **ping** *xxxxx*)。 再現できない場合には、問題の最も単純な例を示してください。
- パフォーマンスの低下は断続的ですか。 パフォーマンスの低下は、一時的ですか。 パフォーマンスの低下は、1 日の特定の時刻に発生しますか、または特定のアクティビティーに関係して発生しますか。

- すべてが遅くなりますか、または特定のものだけが遅くなりますか。
- どの局面が遅いですか。 例えば、文字をエコーする時間、またはトランザクションの完了までの経過時間、または画面をペイントするための時間。
- 問題が発生し始めた時期はいつですか。 システムが最初にインストールまたは実動に入って以来、同じ状態ですか。 問題が発生する前に、システムで何かを変更しましたか (例えば、ユーザーの追加またはシステムへの追加データの移行)。
- クライアント/サーバーの場合、サーバー上でローカルに実行した時に問題は再現されますか (ネットワーク対サーバーの問題)。
- ネットワークに関係している場合、ネットワーク・セグメントはどのように構成されていますか (10 Mb/ 秒または 9600 ボーなどの帯域幅)。 クライアントとサーバー間にルーターが存在していますか。
- どのようなベンダー・アプリケーションがシステム上で実行していますか。また、それらのアプリケーションはパフォーマンス問題に関係していますか。
- パフォーマンス上の問題がユーザーにどのような影響を与えていますか。

パフォーマンス上の問題の報告

オペレーティング・システムのパフォーマンス上の問題は IBM サポート部門に報告する必要があります。標準のソフトウェア問題報告チャネルを使用してください。 ユーザーの組織の正しい問題報告チャネルをご存じない場合には、IBM 担当員にお問い合わせください。

AIX のパフォーマンス上の問題が疑われる場合は、AIX パフォーマンス PMR (perfpmr) データ収集ツールがパフォーマンス・データの収集には最良の方法です。 Web 上の ftp://ftp.software.ibm.com/aix/tools/perftools/perfpmr にあるこれらのツールにアクセスし、測定する AIX バージョンと一致するディレクトリーにある README ファイルの指示に従って、ツールの入手、インストール、およびシステムでのデータ収集を行ってください。 PMR をオープンした後、分析のために IBM にデータを送る方法についても記載されています。

誰かがパフォーマンス上の問題を報告した場合、データを収集し、分析するだけでは不十分です。 パフォーマンス上の問題の性質がわからない場合には、報告している問題と無関係であるデータの分析に、多くの時間を浪費している可能性があります。

サポート担当者に連絡して問題を報告する前に、問題の調査を容易にするために提供を依頼される情報を準備してください。 ユーザーの地域のサポート担当者は、パフォーマンス上の問題の迅速な解決をユーザーと共にすぐに試みます。

問題のより速い解決に役立つ追加の 3 つの方法を以下に示します。

1. 問題の具体的で単純な実例に関する明確な記述を提出します。ただし、症状および事実を、理論、思い付きおよびユーザー自身の結論から必ず分離してください。 「システムが遅い」ことを報告する PMR には、「遅い」が何を意味するか、どのように測定したか、およびどの程度のパフォーマンスであれば受け入れ可能であるかを判別する広範囲の調査が必要です。
2. 問題の前の数週間に、システムに対して行われたすべての変更に関する情報を提出します。 変更したもののうち何かが欠如していると、可能な調査の筋道がブロックされ、解決策の発見が遅れる可能性があります。 すべての事実が準備されていれば、パフォーマンス・チームは無関係の事実を迅速に除去することができます。
3. 正しいマシンを使用して情報を提供してください。 非常に大きなサイトでは、意図せずに、正しくないマシンでデータを収集する可能性があります。 この場合、問題の調査は非常に困難になります。

問題を報告する場合には、以下の基本的な情報を提供してください。

- 問題のヒストリー・データベースを検索して、同様の問題が既に報告されていないかどうかを調べるために使用できる問題記述。
- 問題の原因がオペレーティング・システムの欠陥であるという結論に導いたのは、分析のどの局面ですか。
- 問題が発生しているのは、どのようなハードウェアとソフトウェアの構成ですか。
  - 問題は単一のシステムに限定されていますか。または複数のシステムに影響を与えていますか。
  - 影響を受けたシステムのモデル、メモリー・サイズ、ディスクの数とサイズは何ですか。
  - システムに接続されている LAN および他の通信メディアの種類は何ですか。
  - 構成全体に、他のオペレーティング・システムの構成が組み込まれていますか。
- 問題を経験しているプログラムまたはワークロードの特性は何ですか。
  - **time**、**iostat**、および **vmstat** コマンドによる分析は、CPU 制約または入出力制約であることを示していますか。
  - 影響を受けるシステム (ワークステーション、サーバー、マルチユーザー、またはこれらの組み合わせ) でワークロードが実行中ですか。
- 達成されないパフォーマンス目標は何ですか。
  - 主な目標は、コンソールまたは端末の応答時間、スループット、あるいはリアルタイムの応答性に関係していますか。
  - 目標は、別のシステム上での測定から得られたものですか。 そうであれば、別のシステムはどのような構成ですか。

今回が、この問題に関する最初の報告である場合には、提供する追加のデータを識別するために使用する、また、将来の参照で使用する PMR 番号を受け取ります。

PMR に関するサポート情報および **perfpmr** データを最初に収集する場合、以下のすべての項目を含めてください。

- 問題を再現する方法
  - 可能であれば、問題を再現できるプログラムまたはシェル・スクリプトを含めてください。
  - 最低でも、問題が発生した時の状態に関する詳細な説明が必要です。
- 問題が発見されたアプリケーション:
  - アプリケーションがソフトウェア・プロダクトであるか、ソフトウェア・プロダクトに依存している場合には、そのプロダクトの正確なバージョンとリリースを識別する必要があります。
  - ユーザー作成アプリケーションのソース・コードを提出できない場合には、実行可能プログラムを作成するために使用したコンパイラー・パラメーターの正確なセットを文書化する必要があります。

コマンドとサブルーチンのモニターとチューニング

システムはパフォーマンス関連のコマンドとサブルーチンを提供します。

システム環境用のパフォーマンス・ツールは、何が発生しているかを示すツールと、発生していることに関して何かを行うように指示するツールの、 2 つの一般的なカテゴリーに分類できます。 いくつかのツールは、この両方を行います。 各コマンドの構文と機能の詳細については、コマンド・リファレンスを参照してください。

パフォーマンス関連のコマンドは、基本オペレーティング・システムと共に出荷される、perfagent.tools、 bos.acct、bos.sysmgt.trace、bos.adt.samples, bos.perf.tools、および bos.perf.tune ファイルセットの一部としてパックされています。

次のいずれかのコマンドを実行すれば、すべてのパフォーマンス・ツールがインストールされているか判別できます。

```
# lslpp -lI perfagent.tools bos.sysmgt.trace bos.acct bos.perf.tools bos.perf.tune
```

パフォーマンスの報告および分析コマンド

パフォーマンスの報告および分析コマンドは、システムの 1 つ以上の局面のパフォーマンスに関する情報、あるいはパフォーマンスに影響を与える 1 つ以上のパラメーターに関する情報を提供します。

このようなコマンドを以下に示します。

コマンド
: 機能

alstat 位置合わせ例外統計情報を表示します。

atmstat
: 非同期転送モード (ATM) アダプター統計情報を表示します。

curt 各カーネル・スレッドの CPU 使用率を報告します。

emstat
: エミュレーション命令カウントを報告します。

entstat
: イーサネット・デバイス・ドライバーとデバイス統計情報を表示します。

fddistat
: FDDI のデバイス・ドライバーとデバイスの統計情報を表示します。

filemon
: トレース・ユーティリティーを使用して、物理ボリューム、論理ボリューム、個々のファイル、および仮想メモリー・マネージャーの入出力アクティビティーについて報告します。

fileplace
: 論理または物理ボリューム内のファイル・ロックの配置を表示します。

gprof プログラムのサブルーチン間の制御フローと、各サブルーチンにかかった CPU 時間について報告します。

ifconfig
: TCP/IP を使用するネットワークに関するネットワーク・インターフェース・パラメーターを構成または表示します。

ioo 入出力関連のチューニング・パラメーターを設定します (**vmo** とともに、**vmtune** コマンドを置き換えます)。

iostat 以下のものに関する使用率データを表示します。

- 端末
- CPU
- ディスク
- アダプター

ipfilter
ipreport 出力ファイルから各種の操作ヘッダーを抽出し、それらをテーブルに表示します。

ipreport
指定されたパケット・トレース・ファイルからパケット・トレース・レポートを生成します。

iptrace
インターネット・プロトコルに対してインターフェース・レベルのパケット・トレースを行います。

locktrace
ロック・トレースをオンにします。

lsattr パフォーマンスに影響を与える、以下のようなシステム属性を表示します。

- プロセッサー速度
- キャッシュのサイズ
- 実メモリーのサイズ
- ブロック入出力バッファー・キャッシュ内の最大ページ数
- mbuf に許可されるメモリーの最大 K バイト数
- ディスク入出力ペーシングの最高水準点と最低水準点

lsdev システム内の各デバイスとその特性を表示します。

lslv 論理ボリュームに関する情報を表示します。

lsps ページング・スペースの特性を表示します。

lspv ボリューム・グループ内の物理ボリュームに関する情報を表示します。

lsvg ボリューム・グループに関する情報を表示します。

mtrace
送信元から受信側にマルチキャスト・パスを印刷します。

netpmon
トレース機能を使用して、以下のものを含むネットワーク・アクティビティーを報告します。

- CPU の使用量
- データ転送速度
- 応答時間

netstat
通信アクティビティーに関する、以下のような広範囲の構成情報および統計情報を表示します。

- mbuf プールの現在の状況
- 経路指定テーブル
- ネットワーク・アクティビティーに関する累積統計情報

nfso NFS オプションの値を表示 (または変更) します。

nfsstat
ネットワーク・ファイルシステム (NFS) およびリモート・プロシージャー・コール (RPC) の呼び出しに関する統計情報を表示します。

no 以下のようなネットワーク・オプションの値を表示 (または変更) します。

- デフォルトの送信および受信ソケット・バッファーのサイズ
- mbuf プールおよびクラスター・プールで使用されるメモリーの最大合計量

pdt_config
パフォーマンス診断ツールのパラメーターの始動、停止、または変更を行います。

pdt_report
現在のヒストリー上のデータに基づいて PDT レポートを生成します。

pprof 一定の期間にわたってすべてのカーネル・スレッドの CPU 使用量を報告します。

prof オブジェクト・ファイルのプロファイル・データを表示します。

ps システム内のプロセスに関する、以下のような統計情報および状況情報を表示します。

- プロセス ID
- 入出力アクティビティー
- CPU 使用率

sar 以下のようなオペレーティング・システムのアクティビティーに関する統計情報を表示します。

- ディレクトリー・アクセス
- 読み取りおよび書き込みシステム・コール
- fork および exec
- ページング・アクティビティー

schedo
CPU スケジューラー用のチューニング・パラメーターを設定します (**schedtune** 開始コマンドを置き換えます)。

smitty
システム管理パラメーターを表示 (または変更) します。

splat 競合分析ツールをロックします。

svmon
システム、プロセス、およびセグメントのレベルでのメモリーの状況について報告します。

tcpdump
パケットのヘッダーを出力します。

time、timex
コマンドの実行時間を出力します。CPU 時間を印刷します。

topas 選択されたローカル・システム統計情報を報告します。

tokstat
トークンリング・デバイス・ドライバーとデバイスの統計情報を表示します。

tprof トレース・ユーティリティーを使用して、カーネル・サービス、ライブラリー・サブルーチン、アプリケーション・プログラム・モジュール、およびアプリケーション・プログラム内のソース・コードの個々の行の CPU 使用量について報告します。

trace、trcrpt
システム内のアクティビティーの正確な順序を記録するファイルを作成します。

traceroute
IP パケットのネットワーク・ホストへの経路を出力します。

vmo VMM 関連のチューニング・パラメーターを設定します (**ioo** とともに、**vmtune** コマンドを置き換えます)。

vmstat
以下のような VMM データを表示します。

- ディスパッチ可能または待ち状態であるプロセスの数
- ページ・フレームのフリー・リストのサイズ
- ページ・フォールト・アクティビティー
- CPU 使用率

パフォーマンスのチューニング・コマンド

AIX は、システムのパフォーマンス関連の 1 つ以上の局面を変更することができる幾つかのコマンドをサポートしています。

コマンド
機能

bindprocessor
プロセッサーのカーネル・スレッドをプロセッサーにバインドまたはアンバインドします。

caccelstat
システム全体または各アクセラレーターおよびプロセスの、コヒーレント・アクセラレーターに関連する統計のレポートを作成します。

chdev デバイスの特性を変更します。

chlv 論理ボリュームの特性のみを変更します。

chps ページング・スペースの属性を変更します。

fdpr ユーザー・レベルのアプリケーション・プログラムの実行時間と実メモリー使用率を改善するための、パフォーマンス・チューニング・ユーティリティー。

ifconfig
TCP/IP を使用するネットワークに関するネットワーク・インターフェース・パラメーターを構成または表示します。

ioo 入出力関連のチューニング・パラメーターを設定します (**vmo** とともに、**vmtune** コマンドを置き換えます)。

migratepv
ある物理ボリュームに割り当てられた物理パーティションを、1 つ以上の他の物理ボリュームに移動します。

mkps システムにページング・スペースを追加します。

nfso ネットワーク・ファイルシステム (NFS) ネットワーク変数を構成します。

nice 低いまたは高い優先順位でコマンドを実行します。

no ネットワーク属性を構成します。

renice プロセス実行の nice の値を変更します。

reorgvg
ボリューム・グループの物理パーティション割り当てを再編成します。

rmss アプリケーションのパフォーマンス・テスト用に、さまざまなサイズのメモリーのシステムをシミュレートします。

schedo
CPU スケジューラー用のチューニング・パラメーターを設定します (**schedtune** 開始コマンドを置き換えます)。

smitty
システム管理パラメーターを変更 (または表示) します。

tuncheck
チューニング・パラメーター値を使用してスタンザ・ファイルの妥当性を検査します。

tundefault
すべてのチューニング・パラメーターをデフォルト値にリセットします。

tunrestore
すべてのチューニング・パラメーター値をスタンザ・ファイルから復元します。

tunsave
すべてのチューニング・パラメーター値をスタンザ・ファイルに保存します。

vmo VMM 関連のチューニング・パラメーターを設定します (**ioo** とともに、**vmtune** コマンドを置き換えます)。

関連情報:

caccelstat コマンド

パフォーマンス関連のサブルーチン

AIX は、パフォーマンスのモニターとチューニングで使用することができる幾つかのサブルーチンをサポートしています。

bindprocessor()
カーネル・スレッドをプロセッサーにバインドまたはアンバインドします。

getpri()
実行中のプロセスのスケジューリング優先順位を判別します。

getpriority()
実行中のプロセスの nice の値を判別します。

getrusage()
システム・リソースの使用に関する情報を検索します。

nice() コマンドを低い優先順位あるいは高い優先順位で実行します。

psdanger()
ページング・スペースの使用に関する情報を検索します。

setpri()
実行中のプロセスの優先順位を固定した優先順位に変更します。

setpriority()
実行中のプロセスの nice の値を設定します。

---

ld コマンドの効率的な使用法

バインダー (コンパイルの最終ステージとして起動されるか、または **ld** コマンドによって直接起動します) には、一般的な UNIX リンカーにはない機能があります。

このため、オペレーティング・システムのバインダーの追加の能力を活用しない場合には、リンク時間が長くなる可能性があります。 このセクションでは、バインダーをより効率的に使用するための、いくつかの手法について説明します。

例

ld コマンドの効率的な使用法の例を以下に示します。

1. ライブラリーをプリバインドするには、アーカイブ・ファイルに以下のコマンドを使用してください。

   ```
   # ld -r libfoo.a -o libfooa.o
   ```

2. FORTRAN プログラム `something.f` のコンパイルとバインドには、以下のコマンドを使用します。

   ```
   # xlf something.f libfooa.o
   ```

   プリバインド・ライブラリーは、通常のライブラリー識別構文 **(-lfoo)** を持っていないので、別の通常の入力ファイルとして処理されることに注意してください。

3. バグを修正した後に、モジュールを再コンパイルし、実行可能プログラムを再バインドするには、以下のコマンドを使用します。

   ```
   # xlf something.f a.out
   ```

4. しかし、バグの修正を行った結果、ライブラリー内の別のサブルーチンを呼び出すことになった場合には、バインドが失敗する可能性があります。 以下の Korn シェル・スクリプトが、失敗に関する戻りコードをテストし、リカバリーを行います。

   ```
   # !/usr/bin/ksh
   # Shell script for source file replacement bind
   #
   xlf something.f a.out
   rc=$?
   if [ "$rc" != 0 ]
   then
   echo "New function added ... using libfooa.o"
   xlf something.o libfooa.o
   fi
   ```

再バインド可能な実行可能プログラム

バインダーに関する公式の文書は、入力として実行可能プログラム (ロード・モジュール) を使用できるバインダーの機能について言及しています。

この機能を活用すれば、ソフトウェア開発のワークロードおよび各 **ld** コマンドの応答時間の減少によって、システム全体のパフォーマンスを大幅に改善することができます。

大部分の一般的な UNIX システムでは、 **ld** コマンドは入力として常に、個々の `.o` ファイルからの、または `.o` ファイルのアーカイブ・ライブラリーからのオブジェクト・コードを含んでいるファイルのセットを使用します。 次に、`ld` コマンドは、これらのファイル間の外部参照を解決し、`a.out` というデフォルト名を使用して、実行可能プログラムを作成します。 `a.out` ファイルは実行のみ可能です。 `a.out` ファイルに組み込まれているモジュールのいずれかでバグが検出された場合、欠陥のあるソース・コードを変更し、再コンパイルした後、`.o` ファイルの完全なセットから始めて、**ld** プロセス全体を繰り返す必要があります。

しかし、このオペレーティング・システムのバインダーの場合、解決済みの外部シンボル・ディクショナリー (ESD) と再配置ディクショナリー (RLD) 情報を実行可能ファイルに組み込んでいるので、入力として `.o` と `a.out` の両方のファイルを受け入れることができます。これは、ユーザーが、既存の実行可能プログラムを再バインドして、変更済みの単一の `.o` ファイルを置換できるので、新規の実行可能プログラムを最初から作成せずに済むことを意味します。 バインド・プロセスは、アクセスするさまざまなファイルの

数、および解決する必要があるシンボルへのさまざまな参照の数にある程度まで比例して、ストレージおよびプロセッサー・サイクルを使用するので、実行可能プログラムを 1 つのモジュールの新規バージョンと再バインドすることは、最初からバインドをするのに比べてはるかに迅速です。

プリバインド・サブルーチン・ライブラリー

一部の環境で同様に重要なことは、サブルーチン・ライブラリー全体を、使用する前にバインドしておくという能力です。

`libc.a` のようなシステム・サブルーチン・ライブラリーは、実際に、`.o` ファイルのアーカイブ・ファイルとしてでなく、バインダー出力フォーマットで出荷されます。 これによって、アプリケーションを必要なシステム・ライブラリーにバインドする場合に、アプリケーションからライブラリー・サブルーチンへの参照のみを解決すればよくなるので、ユーザーは処理時間を大幅に減らすことができます。 システム・ライブラリー・ルーチン間の参照は、システム構築プロセス時に解決済みになります。

しかし、多くのサード・パーティーのサブルーチン・ライブラリーが、ローの `.o` ファイルとして、アーカイブ形式で日常的に出荷されています。 ユーザーがアプリケーションをこのようなライブラリーにバインドする場合、バインダーは、アプリケーションのバインドのたびに、ライブラリー全体に関してシンボルを解決する必要があります。 この結果、小さなマシン上でアプリケーションを大きなライブラリーとバインドする環境では、バインド時間が長くなります。

バインド済みライブラリーとバインドされていないライブラリーのパフォーマンス上の差は、最小構成の場合は特に大きくなります。

プロセッサー・タイマーのアクセス

非常に短い時間間隔を測定する試みは、しばしばオペレーティング・システムの一部である断続的なバックグラウンド・アクティビティーによって、またはシステム時間ルーチンが使用する処理時間によって失敗します。 この問題を解決する 1 つの方法は、プロセッサー・タイマーに直接アクセスして測定間隔の開始時刻と終了時刻を判別し、測定を繰り返し実行してから、結果をフィルター処理して、割り込みが介在した期間を除去することです。

POWER および POWER2 のアーキテクチャーは、プロセッサー・タイマーを特殊目的の一対のレジスターとして実装しています。 POWER プロセッサー・ベース アーキテクチャーは、 *TimeBase* という 64 ビットのレジスターを定義しています。 このようなレジスターにアクセスできるのはアセンブラー言語プログラムのみです。

注: プロセッサー・タイマーによって測定される時刻は、絶対的な時刻です。 タイマーへのアクセスと次のアクセスの間に割り込みが発生した場合、計算される期間には、割り込みの処理だけでなく、時間を測定中のコードに制御が戻る前にディスパッチされた他の処理が含まれる可能性があります。 プロセッサー・タイマーからの時間は、ローの時間であり、妥当性テストを受けない状態で使用すべきではありません。

アーキテクチャーに関係なく、3 つのライブラリー・サブルーチンが TimeBase レジスターにアクセスします。これらのサブルーチンは、以下のものです。

read_real_time()
: このサブルーチンは、適切なソースから現在の時刻を取得し、2 つの 32 ビット値として保管します。

read_wall_time()
: このサブルーチンは適切なソースから生の *TimeBase* レジスター値を取得し、2 つの 32 ビット値として保管します。

time_base_to_time()
このサブルーチンは、TimeBase フォーマットから必要な変換を行って、時刻値を秒とナノ秒単位にします。

時刻取得関数と時刻変換関数は、時刻取得のオーバーヘッドを最小化するために分離されています。

以下の例は、コードの特定の部分の経過時間を測定するために、**read_real_time()** および **time_base_to_time()** サブルーチンを使用する方法を示しています。

```
#include <stdio.h>
#include <sys/time.h>

int main(void) {
   timebasestruct_t start, finish;
   int val = 3;
   int w1, w2;
   double time;

   /* get the time before the operation begins */
   read_real_time(&start, TIMEBASE_SZ);

   /* begin code to be timed */
   printf("This is a sample line %d ¥n", val);
   /* end code to be timed   */

   /* get the time after the operation is complete
   read_real_time(&finish, TIMEBASE_SZ);

   /* call the conversion routines unconditionally, to ensure    */
   /* that both values are in seconds and nanoseconds regardless */
   /* of the hardware platform.                                  */
   time_base_to_time(&start, TIMEBASE_SZ);
   time_base_to_time(&finish, TIMEBASE_SZ);

   /* subtract the starting time from the ending time */
   w1 = finish.tb_high - start.tb_high; /* probably zero */
   w2 = finish.tb_low - start.tb_low;

   /* if there was a carry from low-order to high-order during  */
   /* the measurement, we may have to undo it.                  */
   if (w2 < 0) {
      w1--;
      w2 += 1000000000;
   }

   /* convert the net elapsed time to floating point microseconds */
   time = ((double) w2)/1000.0;
   if (w1 > 0)
      time += ((double) w1)*1000000.0;

   printf("Time was %9.3f microseconds ¥n", time);
   exit(0);
}
```

タイマー・ルーチンの呼び出しおよびタイマー・ルーチンからのリターンのオーバーヘッドを最小化するために、非共用ベンチマークのバインドを試みることができます (415 ページの『ダイナミック・リンクおよび静的リンクを使用する時期』を参照してください)。

これが実際のパフォーマンス・ベンチマークの場合は、このコードが繰り返し測定されることになります。多くの連続する繰り返しが合計の時間を測定される場合は、オペレーションの平均時間を計算できますが、割り込み処理または他の無関係なアクティビティーを含んでいる可能性があります。 多くの繰り返しの時間を個別に測定する場合には、個別の時間の妥当性を検査できますが、測定ごとにタイミング・ルーチンの

オーバーヘッドが含まれます。両方の手法を使用して、結果を比較することが望ましい場合があります。いずれにしろ、測定方法を選択する場合に、その目的が何かを考慮する必要があります。

POWER ベースのアーキテクチャーに固有のタイマー・アクセス

POWER family および POWER2 のプロセッサー・アーキテクチャーには、高精度タイマーが入っている 2 つの特殊目的レジスター (上位レジスターと下位レジスター) が含まれています。

注: 以下の記述は、POWER family および POWER2 アーキテクチャー (および 601 プロセッサー・チップ) のみに適用されます。 このコード例は POWER ベース・システムで正しく機能しますが、一部の命令はシミュレートされます。 プロセッサー・タイマーにアクセスする目的は、低いオーバーヘッドで高い精度の時刻を取得することなので、シミュレーションによって結果の有用性は大幅に減少します。

POWER family および POWER2 のプロセッサー・アーキテクチャーの上位レジスターには秒単位の時刻が入り、下位レジスターにはナノ秒単位の小数秒のカウントが入ります。 下位レジスター内の時刻の実際の精度は、モデルに固有の更新頻度によって異なります。

POWER プロセッサー・ベース タイマー・レジスターにアクセスするためのアセンブラー・ルーチン

アセンブラー言語モジュール (timer.s) は、タイマーの上位レジスターと下位レジスターにアクセスするためのルーチン (rtc_upper および rtc_lower) を提供します。

```
          .globl  .rtc_upper
.rtc_upper: mfspr   3,4         # copy RTCU to return register
            br

          .globl  .rtc_lower
.rtc_lower: mfspr   3,5         # copy RTCL to return register
            br
```

秒単位の時刻を提供するための C サブルーチン

second.c モジュールには、上位レジスターと下位レジスターの内容にアクセスするための timer.s ルーチンを呼び出す C ルーチンが含まれています。

このモジュールは、秒単位の時刻を表す倍精度の実数値を戻します。

```
double second()
{
  int ts, tl, tu;

  ts = rtc_upper();     /* seconds                          */
  tl = rtc_lower();     /* nanoseconds                      */
  tu = rtc_upper();     /* Check for a carry from           */
  if (ts != tu)         /* the lower reg to the upper.      */
    tl = rtc_lower();   /* Recover from the race condition. */
  return ( tu + (double)tl/1000000000 );
}
```

サブルーチン **second()** は、 C ルーチンまたは FORTRAN ルーチンから呼び出せます。

注: 最後のシステム・リセットからの時間の長さに応じて、 second.c モジュールはさまざまな精度を出す可能性があります。 リセットからの時間が長くなると、数値の整数秒によって使用される精度のビットの数が多くなります。 この付録の最初の部分で示した手法は、浮動小数点数に変換する前に、経過時間を入手するために必要な減算を実行することによって、この問題を回避します。

PowerPC システムでのタイマー・レジスターへのアクセス

PowerPC アーキテクチャーには、32 ビットの上位フィールドと下位フィールド (TBU と TBL) に論理的に分割される、64 ビットの *TimeBase* レジスターが含まれています。

TimeBase レジスターは、ハードウェアおよびソフトウェアのインプリメンテーションによって異なり、時によって変化する可能性がある頻度で増分されます。 TimeBase から秒への値の変換は、POWER プロセッサー・ベース アーキテクチャーの場合より複雑な作業です。 PowerPC システムで時刻値を入手するには、**read_real_time()**、**read_wall_time()**、および **time_base_to_time()** インターフェースを使用してください。

second サブルーチンの例

プログラムは **second()** サブルーチンを使用できます。

```
#include <stdio.h>
double second();
main()
{
      double t1,t2;

      t1 = second();
      my_favorite_function();
      t2 = second();

     printf("my_favorite_function time: %7.9f¥n",t2 - t1);
      exit();
}
```

second() サブルーチンを使用する FORTRAN プログラムの例 (`main.f`) を以下に示します。

```
      double precision t1
      double precision t2

      t1 = second()
      my_favorite_subroutine()
      t2 = second()
      write(6,11) (t2 - t1)
11    format(f20.12)
      end
```

`main.c` または `main.f` をコンパイルし、使用するには、以下のコマンドを使用してください。

```
xlc -O3 -c second.c timer.s
xlf -O3 -o mainF main.f second.o timer.o
xlc -O3 -o mainC main.c second.o timer.o
```

マイクロプロセッサー速度の判別

このセクションはマイクロプロセッサー速度を判別するためのプロセスの説明です。

ヘルツ (Hz) 単位のプロセッサー速度を取得するには、次のコマンドを入力します。

```
lsattr -E -l proc0 | grep "Processor Speed"
```

前のリリースを使用する場合には、**uname** コマンドを使用してください。 **uname -m** コマンドを実行すると、以下の形式の出力が生成されます。

```
xxyyyyyymmss
```

ここで:

xx 00

yyyyyy
固有の CPU ID

mm モデル ID (マイクロプロセッサー速度の判別に使用する番号)

ss 00 (サブモデル)

uname -m 出力の mm 値と下記の表を相互参照すれば、プロセッサー速度を判別することができます。

モデル ID	マシン・タイプ	プロセッサー速度	アーキテクチャー
02	7015-930	25	Power
10	7013-530	25	Power
10	7016-730	25	Power
11	7013-540	30	Power
14	7013-540	30	Power
18	7013-53H	33	Power
1C	7013-550	41.6	Power
20	7015-930	25	Power
2E	7015-950	41	Power
30	7013-520	20	Power
31	7012-320	20	Power
34	7013-52H	25	Power
35	7012-32H	25	Power
37	7012-340	33	Power
38	7012-350	41	Power
41	7011-220	33	RSC
43	7008-M20	33	Power
43	7008-M2A	33	Power
46	7011-250	66	PowerPC
47	7011-230	45	RSC
48	7009-C10	80	PowerPC
4C		注 1 参照	
57	7012-390	67	Power2
57	7030-3BT	67	Power2
57	9076-SP2 Thin	67	Power2
58	7012-380	59	Power2
58	7030-3AT	59	Power2
59	7012-39H	67	Power2
59	9076-SP2 Thin w/L2	67	Power2
5C	7013-560	50	Power
63	7015-970	50	Power
63	7015-97B	50	Power
64	7015-980	62.5	Power
64	7015-98B	62.5	Power
66	7013-580	62.5	Power
67	7013-570	50	Power
67	7015-R10	50	Power
70	7013-590	66	Power2
70	9076-SP2 Wide	66	Power2
71	7013-58H	55	Power2
72	7013-59H	66	Power2
72	7015-R20	66	Power2
72	9076-SP2 Wide	66	Power2
75	7012-370	62	Power
75	7012-375	62	Power
75	9076-SP1 Thin	62	Power
76	7012-360	50	Power
76	7012-365	50	Power
77	7012-350	41	Power
77	7012-355	41	Power
77	7013-55L	41.6	Power
79	7013-591	77	Power2
79	9076-SP2 Wide	77	Power2
80	7015-990	71.5	Power2
81	7015-R24	71.5	Power2

89	7013-595	135	P2SC
89	9076-SP2 Wide	135	P2SC
94	7012-397	160	P2SC
94	9076-SP2 Thin	160	P2SC
A0	7013-J30	75	PowerPC
A1	7013-J40	112	PowerPC
A3	7015-R30	注 2 参照	PowerPC
A4	7015-R40	注 2 参照	PowerPC
A4	7015-R50	注 2 参照	PowerPC
A4	9076-SP2 High	注 2 参照	PowerPC
A6	7012-G30	注 2 参照	PowerPC
A7	7012-G40	注 2 参照	PowerPC
C0	7024-E20	注 3 参照	PowerPC
C0	7024-E30	注 3 参照	PowerPC
C4	7025-F30	注 3 参照	PowerPC
F0	7007-N40	50	ThinkPad

注:

1. **uname -m** コマンドがモデル ID として 4C を出力するシステムの場合、モデル ID が 4C であるマシンのプロセッサー速度を判別する唯一の方法は、通常は、システム管理サービスへリブートし、システム構成オプションを選択することです。 ただし、場合によっては、以下の表に示すように、**uname -M** コマンドから取得される情報が役立ちます。

uname -M	マシン・タイプ	プロセッサー速度	プロセッサー・アーキテクチャー
IBM,7017-S70	7017-S70	125	RS64
IBM,7017-S7A	7017-S7A	262	RD64-II
IBM,7017-S80	7017-S80	450	RS-III
IBM,7025-F40	7025-F40	166/233	PowerPC
IBM,7025-F50	7025-F50	注 4 参照	PowerPC
IBM,7026-H10	7026-H10	166/233	PowerPC
IBM,7026-H50	7026-H50	注 4 参照	PowerPC
IBM,7026-H70	7026-H70	340	RS64-II
IBM,7042/7043 (ED)	7043-140	166/200/233/332	PowerPC
IBM,7042/7043 (ED)	7043-150	375	PowerPC
IBM,7042/7043 (ED)	7043-240	166/233	PowerPC
IBM,7043-260	7043-260	200	Power3
IBM,7248	7248-100	100	PowerPersonal
IBM,7248	7248-120	120	PowerPersonal
IBM,7248	7248-132	132	PowerPersonal
IBM,9076-270	9076-SP Silver Node	注 4 参照	PowerPC

2. J シリーズ、R シリーズ、および G シリーズの各システムの場合、以下のコマンドを使用すれば、マイクロプロセッサー・カードの FRU 番号から MCA SMP システム内のプロセッサー速度を判別できます。

```
# lscfg -vl cpucard0 | grep FRU
FRU Number    Processor Type    Processor Speed
     E1D           PowerPC 601       75
     C1D           PowerPC 601       75
     C4D           PowerPC 604       112
     E4D           PowerPC 604       112
     X4D           PowerPC 604e      200
```

3. E シリーズおよび F-30 システムの場合には、以下のコマンドを使用してマイクロプロセッサー速度を判別します。

```
# lscfg -vp | pg
```

以下のグループを探してください。

```
procF0                          CPU Card

Part Number.................093H5280
EC Level....................00E76527
Serial Number...............17700008
```

```
FRU Number.................093H2431
Displayable Message.........CPU Card
Device Specific.(PL)........
Device Specific.(ZA)........PS=166,PB=066,PCI=033,NP=001,CL=02,PBH
                                        Z=64467000,PM=2.5,L2=1024
Device Specific.(RM)........10031997 140951 VIC97276
ROS Level and ID............03071997 135048
```

セクション Device Specific.(ZA) 内の、セクション PS= が MHz で表されたプロセッサー速度です。

4. F-50 と H-50 システムおよび SP Silver Node の場合には、以下のコマンドを使用すれば、F-50 システムのプロセッサー速度を判別することができます。

```
# lscfg -vp | more
```

以下のグループを探してください。

```
Orca M5 CPU:
Part Number.................08L1010
EC Level....................E78405
Serial Number...............L209034579
FRU Number..................93H8945
Manufacture ID..............IBM980
Version.....................RS6K
Displayable Message.........OrcaM5 CPU DD1.3
Product Specific.(ZC).......PS=0013c9eb00,PB=0009e4f580,SB=0004f27
                                        ac0,NP=02,PF=461,PV=05,KV=01,CL=1
```

Product Specific.(ZC) を含んでいる行内の、エントリー PS= が 16 進表記のプロセッサー速度です。 これを実際の速度に変換するには、以下の変換式を使用してください。

```
0009E4F580 = 166 MHz
0013C9EB00 = 332 MHz
```

値 PF= は、プロセッサー構成を示します。

```
251 = 1-way 166 MHz
261 = 2-way 166 MHz
451 = 1-way 332 MHz
461 = 2-way 332 MHz
```

各国語サポート : ロケールと速度

各国語サポート (NLS) によって、さまざまな言語環境でオペレーティング・システムを容易に使用できるようになります。 システムから最適なパフォーマンスを取得するために、NLS を理解して使用することはますます重要になっているので、この付録では、NLS について簡単に説明します。

NLS によって、オペレーティング・システムを個々のユーザーの言語および文化的な背景に合わせて調整することができます。 ロケール は、言語と地理的または文化的な要件の特定の組み合わせであり、en_US (アメリカ合衆国で使用される英語) のような複合名によって識別されます。 サポートされるロケールごとに、メッセージ・カタログ、照合値テーブル、およびそのロケールの要件を定義する他の情報からなるセットがあります。システム管理者は、オペレーティング・システムをインストールする時に、インストールするロケール情報を選択することができます。 そうしておけば、個々のユーザーは、*LANG* および *LC_ALL* 変数を変更することによって、各シェルのロケールを制御することができます。

上記の構造体に準拠しない 1 つのロケールは C (または POSIX) ロケールです。 C ロケールは、ユーザーが明示的に別のロケールを選択しない限り、システム・デフォルトのロケールです。 このロケールは、新規に fork されたプロセスが始動するロケールでもあります。 C ロケールでの実行は、オペレーティング・システムにおいて、オリジナルの一言語使用形式の UNIX での実行とほぼ同じです。 C メッセー

ジ・カタログはありません。 その代わりに、カタログからのメッセージの入手を試みるプログラムには、プログラムにコンパイルされるデフォルト・メッセージが戻されます。 **sort** コマンドなどの一部のコマンドは、元の文字セット固有のアルゴリズムに復帰します。

NLS のパフォーマンスは、一般的に以下の 3 つのカテゴリーに分類されます。 コマンドの実行が通常は最も速い C ロケール、次に速い en_US などの単一バイト (ローマ字) ロケール、およびコマンドの実行が最も遅いマルチバイト・ロケールです。

プログラミングに関する考慮事項

ナショナル・ランゲージ・サポートに関しては、幾つかのプログラミングの問題があります。

歴史的に、C 言語は、地域特性のかなりの部分を、バイト (byte) という語と文字 (character) という語を交換して使用できることで示してきました。 例えば、`char foo[10]` と宣言した配列は、 10 バイトの配列です。 しかし、世界のすべての言語が、単一バイトで表せる文字で書かれているわけではありません。 例えば、日本語や中国語の場合、表示する特定のグラフィック文字を識別するために 2 バイト以上が必要です。 したがって、8 ビットのデータである 1 バイトと、単一のグラフィック文字を表すために必要な情報量である 1 文字を区別します。

各ロケールの 2 つの特性は、そのロケールの 1 文字を表すために必要な最大バイト数、および単一の文字が占有する可能性がある出力表示位置の最大数です。 これらの値は、**MB_CUR_MAX** および **MAX_DISP_WIDTH** マクロを使用して入手できます。 両方の値が 1 であれば、このロケールは、バイトと文字の等価性が依然として成立するロケールです。 いずれかの値が 1 より大きい場合には、文字単位の処理を行うプログラム、または使用する表示位置数を常時認識するプログラムは、そのために国際化対応関数を使用する必要があります。

マルチバイトのエンコードは、可変のバイト数の文字から構成されるので、文字の配列として処理することはできません。 各文字が広範囲の処理を受ける必要がある状況で、効率的なコーディングを可能にするために、固定バイト幅のデータ型 **wchar_t** が定義されています。 **wchar_t** の幅は、サポートされている任意の文字エンコードの変換後の形式を入れるのに十分な幅です。したがって、プログラマーは、**wchar_t** の配列を宣言し、従来の `libc.a` 関数の、ワイド文字相当形式を使用すれば、 **char** の配列に使用するのと (ほぼ) 同じロジックを使用して、これらの配列を処理することができます。

残念なことに、テキストを入力したり、ディスクに保管したり、ディスプレイに表示したりするためのマルチバイト形式から **wchar_t** 形式への変換は、計算機の使用に関して非常に高価です。 このような変換は、**wchar_t** 形式の処理効率が、**wchar_t** 形式との間の変換コストに見合うほど重要な場合にのみ、行ってください。

単純化のための規則

プログラマーが、国際化対応関数をほとんど使用せずに多くのプログラムをマルチバイト・ロケールで効率的に実行できるようにする、マルチバイト文字セットの設計に関するいくつかの制約を認識していない場合には、遅いマルチリンガル・アプリケーション・プログラムを作成する可能性があります。

次に例を示します。

- IBM がサポートするすべてのコード・セットでは、文字コード 0x00 から 0x3F が固有であり、ASCII 標準文字をエンコードします。 固有であるという意味は、これらのビットの組み合わせが、マルチバイト文字のいずれかのバイトとして現れないということです。 null 文字はこのセットに入っているので、**strlen()**、**strcpy()**、および **strcat()** 関数は、単一バイト文字列と同様にマルチバイトにも機能します。 プログラマーは、**strlen()** から戻される値が、文字列内のバイト数であり、文字数ではないことに注意してください。

- 同様に、標準の文字列関数 **strchr(foostr, '/')** は、/ (スラッシュ) が固有のコード・ポイント範囲内にあるので、すべてのロケールで正しく機能します。 実際、大部分の標準区切り文字は 0x00 から 0x3F の範囲内にあるので、大部分の構文解析は、国際化対応関数の使用または **wchar_t** 形式への変換なしに、実行することができます。
- 文字列間の比較は、等しいと等しくないという 2 つのクラスに分類されます。標準の **strcmp()** 関数を使用して、比較を実行します。 以下のようなコマンドを実行する場合、

  ```
  if (strcmp(foostr,"a rose") == 0)
  ```

  "a rose" 以外の名前は探せません。このビットのセットのみを探すことになります。 foostr に "a rosE" が含まれていても、一致は検出されません。
- ロケールが定義した照合シーケンスでの文字列の整列を試みる場合に、等しくない比較が発生します。この場合、以下のコマンドを使用し、

  ```
  if (strcoll(foostr,barstr) > 0)
  ```

  各文字に関する照合情報を入手するパフォーマンス費用を負担することになります。
- プログラムを実行すると、プログラムは常に C ロケールで開始されます。 プログラムが、メッセージ・カタログへのアクセスを含む、 1 つ以上の国際化対応関数を使用している場合、以下のコマンドを実行して、

  ```
  setlocale(LC_ALL, "");
  ```

  国際化対応関数を呼び出す前に、親プロセスのロケールに切り替える必要があります。

ロケールの設定

ロケールの設定には **export** コマンドを使用します。

以下のコマンド・シーケンス

```
LANG=C
export LANG
```

は、デフォルト・ロケールを C に設定します (すなわち、*LC_COLLATE* のような特定の変数が、それ以外の値に明示的に設定されない限り、C が使用されます)。

以下のコマンド・シーケンス

```
LC_ALL=C
export LC_ALL
```

は、すべてのロケール変数を、以前の設定値に関係なく、C に強制的に設定します。

ロケール変数の現在の設定に関するレポートを入手するには、locale と入力してください。

チューナブル・パラメーター

オペレーティング・システムのパラメーターには、パフォーマンスに影響を与えるものが多数あります。

パラメーターは、各セクションで英字順に説明します。

環境変数

環境変数には、スレッド・サポートのチューナブル・パラメーターとそのほかのチューナブル・パラメーターの 2 つのタイプがあります。

スレッド・サポートのチューナブル・パラメーター

チューニングできるスレッド・サポート・パラメーターは多数あります。

1. ACT_TIMEOUT

項目　ディスクリプター
目的：　活動化のタイムアウトの秒数をチューニングします。
値：　デフォルト: DEF_ACTOUT。 範囲 : 正整数。
表示：　**echo $ACT_TIMEOUT**

この値は内部でオンにされるので、初期デフォルト値は **echo** コマンドでは表示されません。
変更：　**ACT_TIMEOUT=n export ACT_TIMEOUT**

変更は、このシェル内で即時に有効になります。 変更は、このシェルからログアウトするまで有効です。 永続的な変更を行うには、**ACT_TIMEOUT**=*n* コマンドを /etc/environment ファイルに追加します。
診断：　N/A
チューニング：　N/A

参照: 78 ページの『スレッド環境変数』。

2. AIXTHREAD_COND_DEBUG

項目　ディスクリプター
目的：　デバッガーが使用する条件変数のリストを維持します。
値：　デフォルト : OFF。 範囲 : ON、OFF。
表示：　**echo $AIXTHREAD_COND_DEBUG**

この値は内部でオンにされるので、初期デフォルト値は **echo** コマンドでは表示されません。
変更：　AIXTHREAD_COND_DEBUG={*ON*|*OFF*}export AIXTHREAD_COND_DEBUG

変更は、このシェル内で即時に有効になります。 変更は、このシェルからログアウトするまで有効です。 永続的な変更を行うには、**AIXTHREAD_COND_DEBUG=*{ON|OFF}*** コマンドを /etc/environment ファイルに追加します。
診断：　この変数の設定を **ON** のままにすると、スレッド化されたアプリケーションのデバッグが容易になりますが、多少のオーバーヘッドがかかります。
チューニング：　プログラムに多数のアクティブな条件変数が含まれていて、条件変数の作成と破棄を頻繁に行う場合、これによって、条件変数のリストを維持するためのより多くのオーバーヘッドが発生する可能性があります。 この変数を **OFF** のままにしておくと、リストは使用不可になります。

83 ページの『スレッド・デバッグ・オプション』を参照してください。

3. AIXTHREAD_DISCLAIM_GUARDPAGES

項目　ディスクリプター
目的：　スタック保護ページが放棄されるかどうかを制御します。
値：　デフォルト : OFF。 範囲 : ON、OFF。
表示：　**echo $AIXTHREAD_DISCLAIM_GUARDPAGES**

この値は内部でオンにされるので、初期デフォルト値は **echo** コマンドでは表示されません。
変更：　AIXTHREAD_DISCLAIM_GUARDPAGES={ON|OFF};export AIXTHREAD_DISCLAIM_GUARDPAGES

変更は、このシェル内で即時に有効になります。 変更は、このシェルからログアウトするまで有効です。永続的な変更を行うには、**AIXTHREAD_GUARDPAGES=*n*** コマンドを /etc/environment ファイルに追加します。
診断：　該当なし
チューニング：　保護ページが pthread スタックに使用される場合、**AIXTHREAD_DISCLAIM_GUARDPAGES = ON** を設定すると、pthread が作成されるときに保護ページは放棄されます。 このパラメーターは、スレッド化されたアプリケーションのメモリー占有スペースを削減できます。

78 ページの『スレッド環境変数』を参照してください。

4. AIXTHREAD_ENRUSG

項目　　ディスクリプター
目的：　　pthread リソース収集を使用可能または使用不可にします。
値：　　デフォルト：OFF。 範囲：ON、OFF。
表示：　　**echo $AIXTHREAD_ENRUSG**

この値は内部でオンにされるので、初期デフォルト値は **echo** コマンドでは表示されません。
変更：　　AIXTHREAD_ENRUSG={*ON*|*OFF*}export AIXTHREAD_ENRUSG

変更は、このシェル内で即時に有効になります。変更は、このシェルからログアウトするまで有効です。 永続的な変更を行うには、**AIXTHREAD_ENRUSG=*{ON|OFF}*** コマンドを /etc/environment ファイルに追加します。
診断：　　このパラメーターを **ON** に設定すると、プロセス内のすべての pthread のリソース収集が可能になりますが、多少のオーバーヘッドがかかります。
チューニング：　　N/A

78 ページの『スレッド環境変数』を参照してください。

5. **AIXTHREAD_GUARDPAGES**

項目　　ディスクリプター
目的：　　pthread スタックの最後に追加する保護ページ数を制御します。
値：　　デフォルト: 1 (この 1 はページ数の 10 進値を表し、ページは 4K、64K、などの場合があります)。範囲: n の範囲。
表示：　　**echo $AIXTHREAD_GUARDPAGES**

これは内部でオンにされるので、初期デフォルト値は **echo** コマンドでは表示されません。
変更：　　AIXTHREAD_GUARDPAGES=*n*export AIXTHREAD_GUARDPAGES

変更は、このシェル内で即時に有効になります。 変更は、このシェルからログアウトするまで有効です。 永続的な変更を行うには、**AIXTHREAD_GUARDPAGES=*n*** コマンドを /etc/environment ファイルに追加します。
診断：　　N/A
チューニング：　　N/A

78 ページの『スレッド環境変数』を参照してください。

6. **AIXTHREAD_MINKTHREADS**

項目　　ディスクリプター
目的：　　使用する必要があるカーネル・スレッドの最小数を制御します。
値：　　デフォルト：8。範囲：正整数値。
表示：　　**echo $AIXTHREAD_MINKTHREADS**

これは内部でオンにされるので、初期デフォルト値は **echo** コマンドでは表示されません。
変更：　　**AIXTHREAD_MINKTHREADS=*n*export AIXTHREAD_MINKTHREADS**

変更は、このシェル内で即時に有効になります。 変更は、このシェルからログアウトするまで有効です。 永続的な変更を行うには、**AIXTHREAD_MINKTHREADS =*n*** コマンドを /etc/environment ファイルに追加します。
診断：　　N/A
チューニング：　　ライブラリー・スケジューラーは、この変数に設定された値を下回ったカーネル・スレッドは再利用しません。カーネル・スレッドは、事実上どの時点でも再利用することができます。 一般的に、カーネル・スレッドは、pthread 終了の結果としてターゲットになります。

参照: 83 ページの『プロセス全体のコンテンション有効範囲の変数』。

7. **AIXTHREAD_MNRATIO**

項目　　ディスクリプター
目的：　　ライブラリーのスケール・ファクターを制御します。 この比率は、pthread の作成および終了時に使用されます。
値：　　デフォルト: 8:1 範囲: 2 つの正数値 (*p:k*)、ここで *k* は、*p* 変数に定義された実行可能 pthread 数を処理するために使用する必要があるカーネル・スレッドの数です。
表示：　　**echo $AIXTHREAD_MNRATIO**

これは内部でオンにされるので、初期デフォルト値は **echo** コマンドでは表示されません。
変更：　　**AIXTHREAD_MNRATIO**=*p:k***export AIXTHREAD_MNRATIO**

変更は、このシェル内で即時に有効になります。変更は、このシェルからログアウトするまで有効です。 永続的な変更を行うには、**AIXTHREAD_MNRATIO**=*p:k* コマンドを `/etc/environment` ファイルに追加します。
診断：　　N/A
チューニング：　　非常に多数のスレッドを使用するアプリケーションに役立ちます。 ただし、パフォーマンスを改善する可能性があるという理由で、常時 1:1 の比率をテストします。

参照： 83 ページの『プロセス全体のコンテンション有効範囲の変数』

8. **AIXTHREAD_MUTEX_DEBUG**

項目　　ディスクリプター
目的：　　デバッガーが使用するアクティブな mutex のリストを維持します。
値：　　デフォルト：OFF。 可能な値：ON、OFF。
表示：　　**echo $AIXTHREAD_MUTEX_DEBUG**

これは内部でオンにされるので、初期デフォルト値は **echo** コマンドでは表示されません。
変更：　　**AIXTHREAD_MUTEX_DEBUG**=*{ON|OFF}***export AIXTHREAD_MUTEX_DEBUG**

この変更は直ちに有効になり、ユーザーがこのシェルをログアウトするまで有効です。 永続的な変更を行うには、**AIXTHREAD_MUTEX_DEBUG**=*{ON|OFF}* コマンドを `/etc/environment` ファイルに追加します。
診断：　　この変数を `ON` に設定すると、スレッド化されたアプリケーションのデバッグが容易になりますが、多少のオーバーヘッドがかかります。
チューニング：　　プログラムに多数のアクティブな mutex が含まれていて、mutex の作成と破棄を頻繁に行う場合、これによって mutex のリストを保持するためのより多くのオーバーヘッドが発生する可能性があります。 この変数の `OFF` 設定のままにしておくと、リストは使用不可です。

参照： 83 ページの『スレッド・デバッグ・オプション』

9. **AIXTHREAD_MUTEX_FAST**

項目　　ディスクリプター
目的：　　最適化された mutex ロック・メカニズムを使用可能にします。
値：　　デフォルト：OFF。 可能な値：ON、OFF。
表示：　　**echo $AIXTHREAD_MUTEX_FAST**

これは内部でオンにされるので、初期デフォルト値は **echo** コマンドでは表示されません。
変更：　　**AIXTHREAD_MUTEX_FAST**=*{ON|OFF}***export AIXTHREAD_MUTEX_FAST**

この変更は直ちに有効になり、ユーザーがこのシェルをログアウトするまで有効です。 永続的な変更を行うには、**AIXTHREAD_MUTEX_FAST**=*{ON|OFF}* コマンドを `/etc/environment` ファイルに追加します。
診断：　　この変数を `ON` に設定すると、スレッド化されたアプリケーションに最適化された mutex ロック・メカニズムの使用を強制し、パフォーマンスが改善されます。
チューニング：　　プログラムは過度の mutex コンテンションのためにパフォーマンスが低下することがあります。この場合は、この変数を `ON` に設定して pthread ライブラリーがプロセス専用 mutex のみで作動するように、最適化した mutex のロック・メカニズムを必ず使用します。 これらのプロセス専用 mutex は pthread_mutex_init ルーチンを使用して初期化し、破壊するときは pthread_mutex_destroy ルーチンを使用する必要があります。

参照： 83 ページの『スレッド・デバッグ・オプション』

10. **AIXTHREAD_READ_GUARDPAGES**

項目　　ディスクリプター
目的：　　pthread スタックの最後に追加する保護ページに対する読み取りアクセスを制御します。
値：　　デフォルト：OFF。 範囲：ON、OFF。
表示：　　**echo $AIXTHREAD_READ_GUARDPAGES**

これは内部でオンにされるので、初期デフォルト値は **echo** コマンドでは表示されません。
変更：　　AIXTHREAD_READ_GUARDPAGES={ON|OFF}export AIXTHREAD_GUARDPAGES

変更は、このシェル内で即時に有効になります。 変更は、このシェルからログアウトするまで有効です。 永続的な変更を行うには、**AIXTHREAD_READ_GUARDPAGES={ON|OFF}** コマンドを /etc/environment ファイルに追加します。
診断：　　N/A
チューニング：　　N/A

78 ページの『スレッド環境変数』を参照してください。

11. AIXTHREAD_RWLOCK_DEBUG

項目　　ディスクリプター
目的：　　デバッガーが使用する読み取り/書き込みロックのリストを保持します。
値：　　デフォルト：OFF。 範囲：ON、OFF。
表示：　　**echo $AIXTHREAD_RWLOCK_DEBUG**

これは内部でオンにされるので、初期デフォルト値は **echo** コマンドでは表示されません。
変更：　　**AIXTHREAD_RWLOCK_DEBUG=***{ON|OFF}***export AIXTHREAD_RWLOCK_DEBUG**

変更は、このシェル内で即時に有効になります。 変更は、このシェルからログアウトするまで有効です。 永続的な変更を行うには、**AIXTHREAD_RWLOCK_DEBUG=***{ON|OFF}* コマンドを /etc/environment ファイルに追加します。
診断：　　このパラメーターを ON に設定すると、スレッド化されたアプリケーションのデバッグが容易になりますが、多少のオーバーヘッドがかかります。
チューニング：　　プログラムに多数のアクティブな読み取り/書き込みロックが含まれていて、読み取り/書き込みロックの作成と破棄を頻繁に行う場合、これによって、読み取り/書き込みロックのリストを保持するためのより多くのオーバーヘッドが発生する可能性があります。 この変数を OFF に設定すると、リストが使用不可になります。

参照： 83 ページの『スレッド・デバッグ・オプション』

12. AIXTHREAD_SUSPENDIBLE

項目　　ディスクリプター
目的：　　**pthread_suspend_np** または **pthread_suspend_others_np** ルーチンで、以下のルーチンを使用するアプリケーションのデッドロックを防ぎます。

- pthread_getrusage_np
- pthread_cancel
- pthread_detach
- pthread_join
- pthread_getunique_np
- pthread_join_np
- pthread_setschedparam
- pthread_getschedparam
- pthread_kill

値：　　デフォルト：OFF。 範囲：ON、OFF。
表示：　　**echo $AIXTHREAD_SUSPENDIBLE**

これは内部でオンにされるので、初期デフォルト値は **echo** コマンドでは表示されません。

項目　　ディスクリプター
変更：　　**AIXTHREAD_SUSPENDIBLE**=*{ON|OFF}***export AIXTHREAD_SUSPENDIBLE**

変更は、このシェル内で即時に有効になります。 変更は、このシェルからログアウトするまで有効です。 永続的な変更を行うには、**AIXTHREAD_SUSPENDIBLE**=*{ON|OFF}* コマンドを /etc/environment ファイルに追加します。

診断：　　この変数に関連して、パフォーマンスが少し低下します。

チューニング：　　この変数は、前述の関数が **pthread_suspend_np routine** または **pthread_suspend_others_np** ルーチンで使用される場合のみ使用可能にしてください。

参照： 83 ページの『スレッド・デバッグ・オプション』

13. AIXTHREAD_SCOPE

項目　　ディスクリプター

目的：　　コンテンションの有効範囲を制御します。 P の値はプロセス・ベースのコンテンションの有効範囲 (M:N) を示します。 S の値はシステム・ベースのコンテンションの有効範囲 (1:1) を示します。

値：　　デフォルト：P。可能な値：P または S。

表示：　　**echo $AIXTHREAD_SCOPE**

これは内部でオンにされるので、初期デフォルト値は **echo** コマンドでは表示されません。

変更：　　**AIXTHREAD_SCOPE**=*{P|S}***export AIXTHREAD_SCOPE**

変更は、このシェル内で即時に有効になります。変更は、このシェルからログアウトするまで有効です。 永続的な変更を行うには、**AIXTHREAD_SCOPE**=*{P|S}* コマンドを /etc/environment ファイルに追加します。

診断：　　予想より少ないスレッドがディスパッチされている場合には、システムの有効範囲を試みる必要があります。

チューニング：　　テストの結果、アプリケーションによっては、システム・ベースのコンテンションの有効範囲 (S) を使用した方が、パフォーマンスが改善されます。この環境変数を使用すると、デフォルト属性を用いて作成されたスレッドにのみ影響を与えます。 デフォルト属性は、**pthread_create** に対する *attr* パラメーターが NULL の場合に使用されます。

参照： 78 ページの『スレッド環境変数』

14. AIXTHREAD_SLPRATIO

項目　　ディスクリプター

目的：　　スリープ中のスレッド用に予約する必要があるカーネル・スレッドの数を制御します。

値：　　デフォルト：1:12。 範囲：2 つの正数値 (*k:p*)。ここで k は、p 個のスリープ中の pthread 用に予約する必要があるカーネル・スレッドの数です。

表示：　　**echo $AIXTHREAD_SLPRATIO**

これは内部でオンにされるので、初期デフォルト値は **echo** コマンドでは表示されません。

変更：　　**AIXTHREAD_SLPRATIO**=*k:p***export AIXTHREAD_SLPRATIO**

変更は、このシェル内で即時に有効になります。変更は、このシェルからログアウトするまで有効です。 永続的な変更を行うには、**AIXTHREAD_SLPRATIO**=*k:p* コマンドを /etc/environment ファイルに追加します。

診断：　　N/A

チューニング：　　一般的に、スリープ中の pthread は通常、一度に 1 つずつウェイクアップされるので、サポートするために必要なカーネル・スレッドはより少なくなります。 これにより、カーネル・リソースが節約できます。

参照： 83 ページの『プロセス全体のコンテンション有効範囲の変数』

15. AIXTHREAD_STK=n

項目　　ディスクリプター
目的：　　各 pthread に割り振るべきバイト数 (10 進数)。 この値は **pthread_attr_setstacksize** ルーチンによってオーバーライドすることができます。
値：　　デフォルト: 32 ビット・アプリケーションの場合は 98,304 バイト、64 ビット・アプリケーションの場合は 196,608 バイト。 範囲 : 最も近いページ (現在は 4 096) に切り上げられる、0 から 268 435 455 の範囲の 10 進数の整数値。
表示：　　**echo $AIXTHREAD_STK**

これは内部でオンにされるので、初期デフォルト値は echo コマンドでは表示されません。
変更：　　**AIXTHREAD_STK=sizeexport AIXTHREAD_STK**

変更は、このシェル内で即時に有効になります。変更は、このシェルからログアウトするまで有効です。 永続的な変更を行うには、**AIXTHREAD_STK=size** コマンドを `/etc/environment` ファイルに追加します。
診断：　　障害が起こっているプログラムの分析がスタック・オーバーフローを示している場合、デフォルトのスタック・サイズを大きくすることができます。
チューニング：　　32 ビット・アプリケーションで、32,000 のスレッドの制限に達しようとしているときは、デフォルトのスタック・サイズを小さくする必要があります。

16. AIXTHREAD_AFFINITY

項目　　ディスクリプター
目的：　　pthread 構造、スタック、およびスレッド・ローカル・ストレージの配置を、拡張アフィニティー使用可能化システム上で制御します。
値：　　デフォルト: existing。 範囲: existing、always、attempt。
表示：　　**echo $AIXTHREAD_AFFINITY**

これは内部でオンにされるので、初期デフォルト値は **echo** コマンドでは表示されません。
変更：　　**AIXTHREAD_AFFINITY** ={default|strict|first-touch} **export**

AIXTHREAD_AFFINITY
診断：　　変数「**strict**」を設定すると、スレッドのパフォーマンスは向上します。ただし、それにより、開始時間が余計にかかります。

この変数を 「**default**」に設定すると、以前にバランスした実装を維持します。

この変数を「**first-touch**」に設定すると、実行時の利点と開始時のパフォーマンス負担をバランスさせます。
チューニング：　　スレッドが長時間実行することが予想される場合、この変数を「**strict**」に設定するとパフォーマンスが向上します。 ただし、短時間実行するスレッドが多くある場合は、この変数は「**default**」または「**first touch**」に設定してください。

参照： 78 ページの『スレッド環境変数』

17. MALLOCBUCKETS

項目　　ディスクリプター
目的：　　多数の小さな割り当て要求を出すアプリケーションのパフォーマンスを改善するために、デフォルトのメモリー・アロケーター内でバケット・ベースの拡張機能を使用可能にします。

項目　　ディスクリプター
値：　　**MALLOCTYPE**=buckets

MALLOCBUCKETS=[[**number_of_buckets**:*n* | **bucket_sizing_factor**:*n* | **blocks_per_bucket**:*n* | **bucket_statistics**:[**stdout**|**stderr**|**pathname**]],...]

以下の表は、**MALLOCBUCKETS** のデフォルト値を示しています。

MALLOCBUCKETS のオプション
　　デフォルト値

number_of_buckets[1]
　　16

bucket_sizing_factor (32 ビット**)**[2]
　　32

bucket_sizing_factor (64 ビット**)**[3]
　　64

blocks_per_bucket
　　1024[4]

注:

1. 指定できる最小値は 1 で、最大値は 128 です。

2. 32 ビットのインプリメンテーションの場合、bucket_sizing_factor に指定する値は 8 の倍数でなければなりません。

3. 64 ビットのインプリメンテーションの場合、bucket_sizing_factor に指定する値は 16 の倍数でなければなりません。

4. **bucket_statistics** オプションは、デフォルトでは使用不可です。

表示：　　**echo $MALLOCBUCKETS**; **echo $MALLOCTYPE**
変更：　　環境変数をエクスポートするための、シェル固有の方法を使用します。
診断：　　malloc のパフォーマンスが低く、多くの小さな malloc 要求が出される場合、このフィーチャーによってパフォーマンスを改善することができます。

項目　　ディスクリプター
チューニング：　　malloc バケットを使用可能にするには、*MALLOCTYPE* 環境変数を値 "buckets" に設定する必要があります。

MALLOCBUCKETS 環境変数は、デフォルト値が大部分のアプリケーションに十分なはずですが、malloc バケットのデフォルト構成を変更するために使用することができます。

number_of_buckets:n オプションを使用すれば、ヒープごとに使用可能なバケット数を指定することができます。ここで、*n* はバケット数です。 *n* に指定した値は、使用可能なすべてのヒープに適用されます。

bucket_sizing_factor:n オプションを使用すれば、バケット見積ファクターを指定することができます。ここで、*n* はバイト単位のバケット見積ファクターです。

blocks_per_bucket:n オプションを使用すれば、各バケットに最初に入れるブロック数を指定することができます。ここで、*n* はブロック数です。 この値は、すべてのバケットに適用されます。 *n* の値は、バケットのすべてのブロックが割り当てられた後に、バケットを自動的に拡大するために追加するブロック数を判別するためにも使用されます。

bucket_statistics オプションによって、malloc サブシステムは、malloc バケットが使用可能な間に malloc サブシステムを呼び出す各プロセスの正常終了時に、malloc バケットに関する統計情報の要約を出力します。 この要約は、バケット構成情報およびバケットごとに処理された割り当て要求の数を表示します。 malloc のマルチヒープによって複数のヒープが使用可能になっている場合、バケットごとに示される割り当て要求の数は、すべてのヒープごとに当該バケットに関して処理された、すべての割り当て要求の合計になります。

バケットの統計情報要約は、**bucket_statistics** オプションの指定に従って、以下のいずれかの出力宛先に書き込まれます。

stdout　標準出力

stderr　標準エラー

pathname
ユーザー指定のパス名

ユーザー指定のパス名を提供すると、統計出力は、ファイルの既存の内容がある場合には付加されます。出力が別のプロセスへの入力としてパイピングされるプロセスの場合には、出力宛先として標準出力を使用しないでください。

参照: Malloc バケット

18. MALLOCMULTIHEAP

項目　　ディスクリプター
目的：　　プロセスの専用セグメント内のヒープ数を制御します。
値：　　デフォルト: 32。範囲：正数 1 から 32。
表示：　　**echo $MALLOCMULTIHEAP**

これは内部でオンにされるので、初期デフォルト値は **echo** コマンドでは表示されません。
変更：　　**MALLOCMULTIHEAP**=[[**heaps**:*n* | **considersize**],...] **export MALLOCMULTIHEAP**

変更は、このシェル内で即時に有効になります。変更は、このシェルからログアウトするまで有効です。 永続的な変更を行うには、**MALLOCMULTIHEAP**=[[**heaps**:*n* | **considersize**],...] コマンドを `/etc/environment` ファイルに追加します。
診断：　　malloc ロック (セグメント F にある) 上のロック・コンテンション、または予想より少ない実行可能スレッドを探します。

項目　　ディスクリプター
チューニング：　ヒープの数が少なければ、プロセスのサイズを縮小することができます。 malloc サブシステムを多用する特定のマルチスレッド化ユーザー・プロセスは、アプリケーションを始動する前に **MALLOCMULTIHEAP**=1 環境変数をエクスポートすることにより、パフォーマンスを改善できる場合があります。

パフォーマンスが改善される可能性が特に高いのは、マルチスレッド化 C++ プログラムの場合です。その理由は、これらのプログラムが、コンストラクターまたはデストラクターが呼び出されると必ず malloc サブシステムを使用するからです。

有効なパフォーマンスの改善が最も顕著に認められるのは、マルチスレッド化ユーザー・プロセスを SMP システムで実行中の場合で、特にシステム有効範囲スレッドが使用されている場合です (M:N の比率が 1:1)。 ただし、場合によっては、その他の条件下でも、またユニプロセッサー上でも、明らかな改善が見られることがあります。

considersize オプションを指定すると、代替ヒープ選択アルゴリズムが使用され、要求を処理するために十分なフリー・スペースがあって使用可能であるヒープの選択を試みます。 これにより、**sbrk()** コールの数を減らして、プロセスの作業セットのサイズを最小化することができます。ただし、このアルゴリズムに必要な処理時間は少し長くなります。

参照： 78 ページの『スレッド環境変数』

19. NUM_RUNQ

項目　　ディスクリプター
目的：　実行キューのデフォルト数の数を変更します。
値：　デフォルト：実行時に検出されるアクティブ・プロセッサーの数。 範囲：正整数。
表示：　**echo $NUM_RUNQ**

これは内部でオンにされるので、初期デフォルト値は **echo** コマンドでは表示されません。
変更：　**NUM_RUNQ=n export NUM_RUNQ**

変更は、このシェル内で即時に有効になります。 変更は、このシェルからログアウトするまで有効です。 永続的な変更を行うには、**NUM_RUNQ**=*n* コマンドを `/etc/environment` ファイルに追加します。
診断：　N/A
チューニング：　N/A

参照： 78 ページの『スレッド環境変数』

20. NUM_SPAREVP

項目　　ディスクリプター
目的：　`pth_init` 時間中に malloc される vp 構造の数を設定します。
値：　デフォルト：NUM_SPARE_VP。範囲：正整数。
表示：　**echo $NUM_SPAREVP**

これは内部でオンにされるので、初期デフォルト値は **echo** コマンドでは表示されません。
変更：　**NUM_SPAREVP=n export NUM_SPAREVP**

変更は、このシェル内で即時に有効になります。 変更は、このシェルからログアウトするまで有効です。 永続的な変更を行うには、**NUM_SPAREVP**=*n* コマンドを `/etc/environment` ファイルに追加します。
診断：　N/A
チューニング：　N/A

参照： 78 ページの『スレッド環境変数』

21. SPINLOOPTIME

項目　　ディスクリプター
目的：　　別のプロセッサーに譲る前にビジー・ロックを再試行する回数を制御します (libpthreads の場合のみ)。
値：　　デフォルト：ユニプロセッサーの場合は 1、マルチプロセッサーの場合は 40。 範囲：正整数。
表示：　　**echo $SPINLOOPTIME**

これは内部でオンにされるので、初期デフォルト値は **echo** コマンドでは表示されません。
変更：　　**SPINLOOPTIME=***n***export SPINLOOPTIME**

変更は、このシェル内で即時に有効になります。変更は、このシェルからログアウトするまで有効です。 永続的な変更を行うには、**SPINLOOPTIME=***n* コマンドを /etc/environment ファイルに追加します。
診断：　　スレッドが頻繁にスリープする (アイドル時間が長い) 場合、**SPINLOOPTIME** が十分に大きくない可能性があります。
チューニング：　　pthread mutex コンテンションがある場合、マルチプロセッサー・システムのデフォルト 40 から値を増加すると、パフォーマンスが改善される可能性があります。

参照： 78 ページの『スレッド環境変数』

22. STEP_TIME

項目　　ディスクリプター
目的：　　活動化タイムアウト中に VP を作成する回数をチューニングします。
値：　　デフォルト：DEF_STEPTIME。範囲：正整数。
表示：　　**echo $STEP_TIME**

これは内部でオンにされるので、初期デフォルト値は **echo** コマンドでは表示されません。
変更：　　**STEP_TIME=n export STEP_TIME**

変更は、このシェル内で即時に有効になります。 変更は、このシェルからログアウトするまで有効です。 永続的な変更を行うには、**STEP_TIME=***n* コマンドを /etc/environment ファイルに追加します。
診断：　　N/A
チューニング：　　N/A

参照： 78 ページの『スレッド環境変数』

23. VP_STEALMAX

項目　　ディスクリプター
目的：　　スチールできる VP の数をチューニングまたは VP スチーリングを off にします。
値：　　デフォルト: なし。 範囲：正整数。
表示：　　**echo $VP_STEALMAX**

これは内部でオンにされるので、初期デフォルト値は **echo** コマンドでは表示されません。
変更：　　**VP_STEALMAX=n export VP_STEALMAX**

変更は、このシェル内で即時に有効になります。 変更は、このシェルからログアウトするまで有効です。 永続的な変更を行うには、**VP_STEALMAX=***n* コマンドを /etc/environment ファイルに追加します。
診断：　　N/A
チューニング：　　N/A

参照： 78 ページの『スレッド環境変数』

24. YIELDLOOPTIME

項目　　ディスクリプター
目的：　　ビジー・ロックに関するブロッキングの前に、プロセッサーを譲る回数を制御します (libpthreads の場合のみ)。 プロセッサーは、十分な優先順位を持つ別の実行可能なカーネル・スレッドがあると想定して、別のカーネル・スレッドに譲られます。
値：　　デフォルト: 0。範囲: 正整数値。
表示：　　**echo $YIELDLOOPTIME**

これは内部でオンにされるので、初期デフォルト値は **echo** コマンドでは表示されません。
変更：　　**YIELDLOOPTIME=***n***export YIELDLOOPTIME**

変更は、このシェル内で即時に有効になります。変更は、このシェルからログアウトするまで有効です。 永続的な変更を行うには、**YIELDLOOPTIME=***n* コマンドを `/etc/environment` ファイルに追加します。
診断：　　スレッドが頻繁にスリープする (アイドル時間が長い) 場合、**YIELDLOOPTIME** が十分に大きくない可能性があります。
チューニング：　　ロックを待っている間に、スレッドがスリープしないようにしたければ、デフォルト値 0 から値を増加するとよい場合があります。

参照： 78 ページの『スレッド環境変数』

各種チューナブル・パラメーター

AIX で使用可能な各種パラメーターの幾つかはチューナブルです。

1. AIX_TZCACHE

項目　　ディスクリプター
目的：　　プロセスの期間全体にわたって、*TZ* 変数の固定されたコピーを保管します。
値：　　デフォルト：設定されていません。

可能な値: ON (パラメーターを有効にする)
表示：　　**$AIX_TZCACHE**
変更：　　**export AIX_TZCACHE=ON**

変更は、以後このシェルから開始されるすべてのプロセスに対して有効になります。*TZ* 変数の初期の開始値を常に使用するように、アプリケーションに指示します。アプリケーションによって頻繁にタイム・ゾーンのルックアップが呼び出される場合に、このプロセスによってパフォーマンスが改善されます。例えば、アプリケーションが現地時間を頻繁に検査する場合がこれに当たります。ただし、アプリケーションが既に開始されている場合、*TZ* 変数の変更は認識されません。
診断：　　このパラメーターは、`/etc/environment` ファイル内の汎用システム構成には推奨されません。このパラメーターは、*TZ* 変数を変更しないアプリケーションが、頻繁にタイム・ゾーンの要求を行う場合に使用してください。
チューニング：　　N/A

2. EXTSHM

項目　　ディスクリプター
目的：　　拡張共有メモリー機能をオンにします。
値：　　デフォルト：設定されていません。

可能な値：ON、1SEG、MSEG
表示：　　**echo $EXTSHM**
変更：　　**export EXTSHM**

変更は、このシェル内で即時に有効になります。 変更は、このシェルからログアウトするまで有効です。 永続的な変更を行うには、**EXTSHM=ON**、**EXTSHM=1SEG**、または **EXTSHM=MSEG** コマンドを `/etc/environment` ファイルに追加します。
診断：　　N/A

項目　　ディスクリプター
チューニング：　値を *ON*、*1SEG* または *MSEG* に設定すると、プロセスは 1 バイトの共有メモリー・セグメント (ただし、これは最も近いページに切り上げられます) を割り当てることができます。 これは、11 個のユーザー共有メモリー・セグメントの制限を実質的に除去します。 32 ビット・プロセスの場合、すべてのメモリー・セグメントの最大サイズは 2.75 GB です。

EXTSHM を *ON* に設定すると、この変数を *1SEG* に設定した場合と同じ効果があります。 いずれの設定でも 256 MB より小さい共有メモリーが mmap セグメントとして内部に作成され、パフォーマンスへの影響は mmap と同じです。 作業セグメントには 256 MB 以上の共有メモリーが内部に作成されます。

EXTSHM を *MSEG* に設定すると、すべての共有メモリーは mmap セグメントとして内部に作成され、メモリーの使用率を向上させます。

参照： 173 ページの『拡張共有メモリー』

3. **LDR_CNTRL**

項目　　ディスクリプター
目的：　カーネル・ローダーのチューニングを可能にします。
値：　デフォルト：設定されていません。

可能な値: PREREAD_SHLIB、LOADPUBLIC、IGNOREUNLOAD、USERREGS、MAXDATA、MAXDATA32、MAXDATA64、DSA、PRIVSEG_LOADS、DATA_START_STAGGER、LARGE_PAGE_TEXT、LARGE_PAGE_DATA、HUGE_EXEC、NAMEDSHLIB、SHARED_SYMTAB、または SED
表示：　**echo $LDR_CNTRL**
変更：　**LDR_CNTRL=**{*PREREAD_SHLIB* | *LOADPUBLIC*| ...}**export LDR_CNTRL** 変更はこのシェルで即時に有効になります。 変更は、このシェルからログアウトするまで有効です。 永続的な変更を行うには、`/etc/environment` ファイルに次の行を追加します。**LDR_CNTRL=**{*PREREAD_SHLIB* | *LOADPUBLIC*| ...}
診断：　N/A

項目	ディスクリプター
チューニング：	**LDR_CNTRL** 環境変数を使用すると、システム・ローダーの動作を 1 つ以上の面で制御できます。 **LDR_CNTRL** 変数には、複数のオプションを指定できます。このオプションを指定する場合、アットマーク符号 (@) を使用してオプションを区切ります。 **LDR_CNTRL**=PREREAD_SHLIB@LOADPUBLIC は複数オプションの指定例です。 PREREAD_SHLIB オプションを指定すると、ライブラリーにアクセスしたときすぐに、ライブラリー全体が読み取られます。 VMM 先読みをチューニングすれば、プログラムがページへのアクセスを開始する前に、ライブラリーをディスクから読み取り、メモリー・キャッシュに入れることができます。 この方法を使用するとメモリー使用量が増えますが、共用ライブラリー・ページを多数使用する、アクセス・パターンが順次でないプログラム (例えば Catia) のパフォーマンスも改善できます。 LOADPUBLIC オプションを指定すると、システム・ローダーは、アプリケーションの要求するモジュールをすべてグローバル共用ライブラリー・セグメントにロードします。 共通のグローバル共用ライブラリー・セグメントにモジュールをロードできない場合、モジュールはアプリケーション専用にロードされます。 IGNOREUNLOAD オプションを指定すると、アプリケーションはライブラリーをアンロードできなくなります。この指定により、メモリーをフラグメント化できなくなり、ライブラリーを繰り返しロードおよびアンロードするときに発生するオーバーヘッドはなくなります。 IGNOREUNLOAD オプションを指定しない場合、アプリケーション・ロード時にモジュールがロードされ、さらにそのモジュールを複数回動的にロードおよびアンロードするよう要求されると、同じモジュールのデータ・インスタンスが 2 つできてしまうこともあります。 USERREGS オプションを指定すると、アプリケーションによって行われるシステム・コール全体で、汎用ユーザー・レジスターがシステムによってすべて保管されます。 このオプションは、ガーベッジ・コレクションを行うアプリケーションに有用です。 MAXDATA オプションを指定すると、プロセスの最大ヒープ・サイズが設定され、実行可能モジュールに指定したすべての maxdata 値の指定変更も行われます。 maxdata 値は、プロセスの初期ソフト・データ・リソース制限の設定に使用されます。 32 ビット・プログラムの場合、maxdata 値を非ゼロにすると、大規模アドレス・スペース・モデルが使用可能になります。Large Program Support を参照してください。 大規模アドレス・スペース・モデルを使用不可にするには、**LDR_CNTRL=MAXDATA=0** の設定により、0 の maxdata 値を指定します。 64 ビット・プログラムの場合、maxdata 値は、プログラムのデータ・ヒープに保証される最大サイズを提供します。 ヒープ用に予約されたアドレス・スペースの部分は、アドレスを明示的に指定した場合でも、**shmat()** サブルーチンまたは **mmap()** サブルーチンでは使用できません。 任意の値を指定できますが、指定された maxdata 値にかかわらず、データ域は 0x06FFFFFFFFFFFFFF を超える拡張はできません。 プロセスが 32 ビットか 64 ビットに基づいて、よりきめ細かい制御を可能にするために、2 つの追加の maxdata オプションが存在します。 これらの追加の maxdata オプションは、対応するオブジェクト・モードの MAXDATA オプションをオーバーライドします。 MAXDATA32 オプションを指定すると、64 ビット・プロセスに対して値が無視されることを除いて、MAXDATA と同じ動作になります。 MAXDATA64 オプションを指定すると、32 ビット・プロセスに対して値が無視されることを除いて、MAXDATA と同じ動作になります。 PRIVSEG_LOADS オプションを指定すると、動的にロードされた専用モジュールをプロセス専用セグメントに書き込むようシステム・ローダーに指図が行われます。 この指定により、専用動的ロードを実行してプロセス・ヒープでメモリー不足になりがちな大容量メモリー・モデル・アプリケーションで、メモリーの可用性が改善される場合があります。 プロセス専用セグメントに十分なスペースがない場合は、PRIVSEG_LOADS オプションを指定しても効果はありません。 PRIVSEG_LOADS オプションは、ゼロ以外の MAXDATA 値を指定した 32 ビット・アプリケーションに対してのみ有効です。 DATA_START_STAGGER=Y オプションを指定すると、プロセスのデータ・セクションは MCM 当たりのオフセットで開始します。このオフセットは **vmo** コマンドの **data_stagger_interval** オプションで制御されます。 指定された MCM で実行される n 番目のラージ・ページ・データ処理には、オフセット (n * **data_stagger_interval** * PAGESIZE) % 16 MB で開始するデータ・セクションがあります。 DATA_START_STAGGER=Y オプションは 64 ビット・カーネル上の 64 ビット・プロセスにのみ有効です。 LARGE_PAGE_TEXT=Y オプションは、ローダーがプロセスのテキスト・セグメントにラージ・ページの使用を試みることを示しています。 LARGE_PAGE_TEXT=Y オプションは 64 ビット・カーネル上の 64 ビット・プロセスにのみ有効です。

項目　　ディスクリプター

LARGE_PAGE_DATA=M オプションを指定すると、データ・セグメント全体が割り振られるのではなく、データ・セグメントのラージ・ページが最大で **brk** 値まで十分なだけ割り振られます。LARGE_PAGE_DATA=M オプションを指定しない場合には、データ・セグメント全体が割り振られます。 **brk** 値への変更をサポートするだけの十分なページがないのに **brk** 値を変更すると、障害が起きる場合があります。

RESOLVEALL オプションを指定すると、ローダーは、プログラム・ロード時またはプログラムが動的モジュールをロードするときにインポートされるすべての未定義シンボルを解決することを強制されます。 シンボルの解決は、AIX の標準の深さ優先順で行われます。 **LDR_CNTRL**=RESOLVEALL を指定した場合にインポートされたシンボルが解決されないときは、そのプログラムまたは動的モジュールのロードは失敗します。

HUGE_EXEC オプションを指定することにより、特定の 32 ビットの実行可能ファイルの読み取り専用セグメントのプロセス・アドレス・スペース・ロケーションに対するユーザー制御が提供されます。詳細情報については、『32 ビット・ヒュージ実行可能ファイル』を参照してください。

NAMEDSHLIB=name,[attr1],[attr2]...[attrN] オプションを指定すると、指定した名前で識別される共用ライブラリー領域をアクセスまたは作成するプロセスが使用可能になります。 名前を付けた共用ライブラリー領域は、以下の方法で作成できます。

- 属性なし
- **doubletext32** 属性あり。この属性は、共用ライブラリー・テキストに専用の 2 つのセグメントを持つ、名前付きの共用ライブラリー領域を作成します

プロセスで、存在しない名前付きの共用ライブラリー領域を使用するよう要求された場合、共用ライブラリー領域は、指定された名前で自動的に作成されます。 無効な名前が指定されている場合、NAMEDSHLIB=name,[attr1],[attr2]...[attrN] オプションは無視されます。 有効な名前は、正の長さのもので、英数字、下線文字、およびピリオド文字が含まれるものです。

SHARED_SYMTAB=Y オプションを指定すると、システムは、プログラムがシンボルをエクスポートする場合、64 ビット・プログラムの共用シンボル・テーブルを作成します。 プログラムの複数のインスタンスが同時に実行される場合、共用シンボル・テーブルを使用すると、プログラムで必要なシステム・メモリー量を削減することができます。

SHARED_SYMTAB=N オプションを指定すると、システムは 64 ビット・プログラムの共用シンボル・テーブルを作成しません。 このオプションは、XCOFF 補助ヘッダー内の AOUT_SHR_SYMTAB フラグをオーバーライドします。

SED オプションを指定すると、実行可能モジュールで指定される他の SED モードを無視することによって、プロセスのスタック実行不可 (SED) モードが設定されます。このオプションは、以下のいずれかの値に設定しなければなりません。

```
SED=system
SED=request
SED=exempt
```

4. LDR_PRELOAD LDR_PRELOAD64

項目　　ディスクリプター

目的 :　　共用ライブラリーのプリロードを要求します。 LDR_PRELOAD オプションは 32 ビット・プロセス用、LDR_PRELOAD64 オプションは 64 ビット・プロセス用です。 シンボルの解決では、この変数にリストされたプリロード・ライブラリーが、最初にすべてのインポートされたシンボルごとに検索され、これらのライブラリーで検出されない場合のみ、通常の検索が使用されます。 プリロードされたライブラリーからのシンボルの先取りは、AIX のデフォルト・リンクとランタイム・リンクの両方で有効です。 据え置きシンボル解決は未変更です。

値 :　　デフォルト : 設定されていません。

可能な値: ライブラリー名 (複数可)

注: 複数のライブラリーがリストされる場合は、コロン (:) で区切ります。 括弧内にアーカイブ・ライブラリーのメンバーを入れます。

表示 :　　**echo $LDR_PRELOAD**

echo $LDR_PRELOAD64

項目　　ディスクリプター
変更：　　**$LDR_PRELOAD="libx.so:liby.a(shr.o)"**

最初に `libx.so` 共用オブジェクトから必要なシンボルを解決し、続いて `liby.a` の `shr.o` メンバーから必要なシンボルを解決し、最後に、プロセスの依存関係内の必要なシンボルを解決します。 さらに、動的にロードされたすべてのモジュール (dlopen() または load() を使用してロードされたモジュール) は、最初に、変数ごとにリストされたプリロード・ライブラリーから解決されます。

診断：　　N/A

5. NODISCLAIM

項目　　ディスクリプター
目的：　　**free()** に対するコールの処理方法を制御します。 **PSALLOC** が *early* に設定されている場合、すべての **free()** コールの結果として、**disclaim()** システム・コールが行われます。 **NODISCLAIM** が *true* に設定されている場合には、このことは起こりません。
値：　　デフォルト：設定されていません。

可能な値：True
表示：　　**echo $NODISCLAIM**
変更：　　**NODISCLAIM=true export NODISCLAIM**

変更は、このシェル内で即時に有効になります。 変更は、このシェルからログアウトするまで有効です。 永続的な変更を行うには、**NODISCLAIM=true** コマンドを `/etc/environment` ファイルに追加します。

診断：　　**disclaim()** システム・コールの数が非常に多い場合には、この変数を設定しなければならない可能性があります。
チューニング：　　**PSALLOC** が *early* に設定されている場合に、この変数を設定すれば、**free()** から **disclaim()** オプションへのコールが除去されます。

参照： 169 ページの『早期ページ・スペース割り当て』

6. NSORDER

項目　　ディスクリプター
目的：　　セット・ネーム・レゾリューションの検索順を上書きします。
値：　　デフォルト：bind、nis、local

可能な値：bind、local、nis、bind4、bind6、local4、local6、nis4、または nis6
表示：　　**echo $NSORDER**

これは内部でオンにされるので、初期デフォルト値は **echo** コマンドでは表示されません。
変更：　　**NSORDER=***value*, *value*, ... **export NSORDER**

変更は、このシェル内で即時に有効になります。 変更は、このシェルからログアウトするまで有効です。 永続的な変更を行うには、**NSORDER=***value* コマンドを `/etc/environment` ファイルに追加します。

診断：　　N/A
チューニング：　　**NSORDER** は、`/etc/netsvc.conf` ファイルをオーバーライドします。

参照： 321 ページの『ネーム・レゾリューションのチューニング』

7. PSALLOC

項目　　ディスクリプター
目的：　　ページング・スペース割り振りポリシーを判別できるように、**PSALLOC** 環境変数を設定します。
値：　　デフォルト：設定されていません。

可能な値：early
表示：　　**echo $PSALLOC**
変更：　　**PSALLOC=early export PSALLOC**

変更は、このシェル内で即時に有効になります。 変更は、このシェルからログアウトするまで有効です。
診断：　　N/A
チューニング：　　低いページング条件のためにプロセスが削除されないように、このプロセスは、早期ページ・スペース割り振りポリシーを使用して、ページング・スペースを事前割り当てすることができます。 しかし、これはむだなページング・スペースを生む可能性があります。 **NODISCLAIM** 環境変数も設定しなければならない場合があります。

参照： 57 ページの『ページング・スペース・スロットの割り当てと再利用』および 169 ページの『早期ページ・スペース割り当て』

8. **RT_GRQ**

項目　　ディスクリプター
目的：　　スレッドを、CPU ごとの実行キューではなく、グローバル実行キューに置きます。
値：　　デフォルト：設定されていません。

範囲：ON、OFF
表示：　　**echo $RT_GRQ**
変更：　　**RT_GRQ=***{OFF/ON***export RT_GRQ**

変更は即時に有効になります。 変更は次のブートまで有効です。 永続的な変更を行うには、**RT_GRQ**=*{ON|OFF}* コマンドを `/etc/environment` ファイルに追加します。
診断：　　N/A
チューニング：　　マルチプロセッサー・システムでチューニングすることができます。 この変数を *ON* に設定すると、スレッドはグローバル実行キューに置かれます。 その場合、どのスレッドが最高の優先順位を持っているかを確かめるために、グローバル実行キューの検索が行われます。 これにより、スレッドがより早くディスパッチされ、SCHED_OTHER を実行中であり、かつ、割り込み駆動型のスレッドのパフォーマンスを改善することができます。

参照： 47 ページの『スケジューラー実行キュー』

9. **RT_MPC**

項目　　ディスクリプター
目的：　　リアルタイム・モードでカーネルを実行中であり (**bosdebug** コマンドを参照)、より高い優先順位のスレッドが実行可能であって、そのスレッドを即時にディスパッチできる場合には、MPC を別の CPU に送信して割り込むことができます。
値：　　デフォルト：設定されていません。

範囲：ON
表示：　　**echo $RT_MPC**
変更：　　**RT_MPC=ON export RT_MPC**

変更は即時に有効になります。 変更は次のブートまで有効です。 永続的な変更を行うには、 **RT_MPC=ON** コマンドを `/etc/environment` ファイルに追加します。
診断：　　N/A

10. **TZ**

項目　　ディスクリプター
目的：　　タイムゾーンを設定します。
値：　　デフォルト: Olson タイムゾーン

可能な値: Olson タイムゾーンまたは POSIX タイムゾーン
表示：　　**echo $TZ**
変更：　　**TZ = value export TZ**

変更は、シェル内で即時に有効になります。 変更は、ユーザーがこのシェルをログアウトするまで有効です。永続的な変更を行うには、**TZ= value** コマンドを /etc/environment ファイルに追加します。
診断：　　N/A
チューニング：　　POSIX を使用する可能性があるアプリケーションは、パフォーマンスが重要であり、タイムゾーン規則や夏時間調整時間の正確な変更に依存しないアプリケーションです。

11. VMM_CNTRL

項目　　ディスクリプター
目的：　　仮想メモリー・マネージャーのチューニングを可能にします。
値：　　デフォルト ：設定されていません。

指定可能な値: vmm_fork_policy、ESID_ALLOCATOR、SHM_1TB_SHARED、SHM_1TB_UNSHARED
表示：　　**echo $ VMM_CNTRL**
変更：　　**VMM_CNTRL={vmm_fork_policy=... | ESID_ALLOCATOR=... | ...}export VMM_CNTRL**

変更は、このシェル内で即時に有効になります。 変更は、ユーザーがこのシェルをログアウトするまで有効です。永続的な変更を行うには、VMM_CNTRL= 環境変数を /etc/environment ファイルに追加します。
診断：　　N/A
チューニング：　　VMM_CNTRL 環境変数を使用すると、仮想メモリー・マネージャーを制御できます。VMM_CNTRL 環境変数を使用して、各オプションを「@」記号で区切ることにより、複数のオプションを指定することができます。複数のオプションを指定する例を次に示します。

```
VMM_CNTRL=vmm_fork_policy=COW@SHM_1TB_SHARED=5
```

vmm_fork_policy=COW オプションを指定すると、プロセスが fork されるたびに vmm はコピー・オン・ライト fork-tree ポリシーを使用します。これはデフォルトの動作です。vmm がコピー・オン・ライト・ポリシーを使用しないようにするには、vmm_fork_policy=COR オプションを使用します。vmm_fork_policy オプションを指定すると、vmm_fork_policy グローバル・チューナブルは無視されます。

ESID_ALLOCATOR オプションを指定すると、アロケーターによる shmat および mmap の宛先のない割り振りが制御されます。詳しくは、 174 ページの『1 TB セグメントの別名割り当て』を参照してください。

SHM_1TB_SHARED または SHM_1TB_UNSHARED を指定すると、1 TB 共有メモリー領域の使用が制御されます。詳しくは、 174 ページの『1 TB セグメントの別名割り当て』を参照してください。

12. AIX_STDBUFSZ

項目　　ディスクリプター
目的：　　**cp**、**mv**、**cat**、**cpio** の各コマンドによって生成される、読み取りと書き込みのシステム呼び出し用の入出力バッファー・サイズを構成します。これは、ストリーム・バッファリングにも適用できます。
値：　　デフォルト: 設定されていません。

可能な値: バイト、KB、MB の単位でバッファー・サイズを指定する整数値。
表示：　　**echo $ AIX_STDBUFSZ**
変更：　　**AIX_STDBUFSZ=1024; export AIX_STDBUFSZ** (バッファー・サイズ 1024 を構成する場合)

変更は、このシェル内で即時に有効になります。変更は、ユーザーがこのシェルをログアウトするまで有効です。バッファー・サイズの永続的な変更を行うには、AIX_STDBUFSZ 環境変数を /etc/environment ファイルに追加します。
診断：　　N/A

項目　　ディスクリプター
チューニング：　　値は次の方法で指定します。

- 整数値は、export AIX_STDBUFSZ=1024 の形式で指定します。
- 16 進数値は、export AIX_STDBUFSZ=0x400 の形式で指定します。
- 制限: 最小限度は 64 バイトで、最大限度は 127 MB です。
- これらの限度の範囲外にある有効な整数は、最も近い限界値に戻されます。
- 指定値が 2 のべき乗ではない場合は、2 のべき乗で最も近い、指定値より小さな値に切り捨てられます。
- AIX_STDBUFSZ パラメーターの値が無効の場合、値は無視されます。

13. **AIX_LDSYM**

項目　　ディスクリプター
目的：　　Lightweight_core ファイル内のソース行情報は、テキスト・ページのサイズが 64 K の場合、デフォルトでは表示されません。テキスト・ページのサイズが 64 K の場合は、Lightweight_core ファイル内のソース行情報を取得するのに、環境変数 AIX_LDSYM=ON を使用してください。
値：　　デフォルト: 設定されていません。

可能な値: ON。
表示：　　**echo $ AIX_LDSYM**
変更：　　**export AIX_LDSYM=ON**

変更は、このシェル内で即時に有効になります。変更は、ユーザーがこのシェルをログアウトするまで有効です。システムの永続的な変更を行うには、AIX_LDSYM=ON 環境変数を /etc/environment ファイルに追加します。
診断：　　N/A
チューニング：　　このパラメーターは、テキスト・ページ・サイズが 64 K で、その Lighweight_core ファイル内のソース行情報を必要とするアプリケーションの場合に使用します。

32 ビットのヒュージ実行可能ファイル

テキストとローダーのセクションがファイルの最初の 256 MB に常駐するほとんどの 32 ビット実行可能ファイルの場合、AIX はプロセス・アドレス・スペースのセグメント 0x1 を実行可能ファイルの読み取り専用情報用に予約します。

ただし、テキストとローダーのセクションを結合したサイズが 1 つのセグメントより大きい 32 ビットのヒュージ実行可能ファイルの場合、複数の連続する読み取り専用セグメントが必要です。

ヒュージ実行可能ファイルの読み取り専用セグメントのプロセス・アドレス・スペース・ロケーション:

LDR_CNTRL 環境変数の **HUGE_EXEC** オプションは、読み取り専用セグメントのプロセス・アドレス・スペース・ロケーションに対するユーザー制御を提供します。

このオプションの使用法は次のとおりです。

LDR_CNTRL=[...@]HUGE_EXEC={<*segno*>|0}[,<*attribute*>][@...]

ここで、*segno* は 32 ビット・プロセス・アドレス・スペース内の要求された開始セグメント番号、またはゼロです。

ゼロ以外の segno 値を指定すると、システム・ローダーはヒュージ実行可能ファイルの読み取り専用セグメントを、要求された開始セグメント番号に対応するロケーションのプロセス・アドレス・スペースに挿入しようとします。

ゼロの segno 値を指定すると、システム・ローダーはヒュージ実行可能ファイルの読み取り専用セグメントを、要求された開始セグメント番号に対応するロケーションのプロセス・アドレス・スペースに挿入しようとします。

ゼロの segno 値を指定すると (または **LDR_CNTRL** に **HUGE_EXEC** オプションを指定しないと)、システム・ローダーはアドレス・スペース・モデルを基に開始セグメント番号を選択します。 この選択を行うために使用されるアルゴリズムは、**mmap** サブルーチンの **MAP_VARIABLE** フラグに似ています。

- プロセスによって Dynamic Segment Allocation (DSA) またはラージ・ページ・データのいずれも要求されていない場合、システムはプロセス・ヒープの直後にある連続セグメントのセットを選択します。
- そうでない場合、システムは一番下にある共用ライブラリー域セグメントのすぐ下にある連続セグメントのセットを選択します。

開始セグメント番号は、要求されたプロセス・アドレス・スペース・モデルによって既に予約されているどのセグメントとも対立しないようにする必要があります。 そのような対立が存在するかどうかを判別するには、プロセス・ヒープと共用ライブラリー域セグメント (ある場合) を考慮する必要があります。 プロセス・ヒープ・セグメントが動的に割り当てられている (DSA またはラージ・ページ・データである) 場合、初期ヒープ・セグメントのみが予約を考慮されます。 選択された開始セグメント番号が予約済みのセグメントと対立した場合は、実行は失敗し、**ENOMEM** が戻されます。

共用ライブラリー・テキストに対するセグメント **0x1** の可用性**:**

ヒュージ実行可能ファイルは複数の連続する読み取り専用セグメントからなり、部分的にもセグメント 0x1 に常駐することはできません。 ヒュージ実行可能ファイルはそれ自体でセグメント 0x1 を使用することができないため、プロセス・アドレス・スペースのこの部分を他の目的のために使用できます。

HUGE_EXEC ローダー制御オプションの任意指定の属性により、共用ライブラリー・テキスト・セグメントを 0xD ではなく 0x1 に配置するように要求することができます。

```
HUGE_EXEC={<segno>|0},shtext_in_one
```

共用ライブラリー域の事前再配置済みデータは共用テキスト・セグメントがセグメント 0xD に常駐する場合にのみ有用であるため、このオプションを要求するプロセスは事前再配置済みライブラリー・データの利益を得ることはできません。 したがって、共用ライブラリー・データはすべてこのプロセス・ヒープに常駐します。 これには、プロセス・ヒープ (mmap/shmat) および実行可能ファイルによる分割使用のためにすべてのセグメント 0x3–0xF を解放するという利点があります。

注**:** 通常プロセスが共用ライブラリー域を使用するのを防ぐ **maxdata** および DSA 設定 (例えば、`maxdata>0xA0000000/dsa` または `maxdata=0/dsa`) とともに使用される **shtext_in_one** 属性により、プロセスは共用ライブラリー・テキストの提供するパフォーマンス上の利点を利用することができます。

プロセスの共用ライブラリー域が **doubletext32** 属性によって作成された名前付き領域の場合は、事前再配置済みデータ・セグメントはなく、両方の共用ライブラリー・テキスト・セグメントを使用する必要があります。 この場合、基本セグメント (通常セグメント 0xD に配置されている) はセグメント 0x1 に移動され、2 次共用ライブラリー・テキスト・セグメントはセグメント 0xF に留まります。 これにより、プロセス・ヒープ (mmap/shmat) および実行可能ファイルが使用するために分割できる連続セグメント (0x3–0xE) の数が最大化されます。

maxdata 値が 0xA0000000 を超えており、DSA が使用可能になっている非ヒュージ実行可能ファイルはいかなる場合も共用ライブラリー域の使用を阻止されますが、一方 (1) **doubletext32** 属性を使用して作成

された名前付き共用ライブラリー域を使用し、(2) **shtext_in_one** 属性を指定するヒュージ実行可能ファイルは、その領域へのアクセス可能性を没収する前に最大 0xC0000000 の **maxdata** 値を要求することができます。

ヒュージ実行可能ファイルの例**:**

ヒュージ実行可能ファイルの使用例のシナリオ

大容量のプログラム・アドレス・スペース・モデルの例

優先アドレス・スペース・モデルが次の場合:

- MAXDATA=0x50000000 (非 DSA)
- セグメント 0xB、0xC および 0xE に必要な shmat/mmap 領域
- 共用ライブラリー域テキストおよび事前再配置済みデータのアクセシビリティー

アドレス・スペースのレイアウトは、次のようになります。

```
0x0: System Segment
0x1:
0x2: Exec-time Private Dependencies / Stack
0x3: Process Heap
0x4: Process Heap
0x5: Process Heap
0x6: Process Heap
0x7: Process Heap
0x8:
0x9:
0xA:
0xB: shmat/mmap (mapped after exec)
0xC: shmat/mmap (mapped after exec)
0xD: Shared Library Text
0xE: shmat/mmap (mapped after exec)
0xF: Pre-relocated Library Data
```

この例から、セグメント 0x8–0xA が実行可能ファイル用に使用できることが分かります。

実行可能ファイルのサイズが 256 MB を超え、512 MB 未満であると仮定した場合、この状態に理想的な **HUGE_EXEC** 設定は次のとおりです。

1. HUGE_EXEC=0
2. HUGE_EXEC=0x8
3. HUGE_EXEC=0x9

オプション 1 と 2 は実行可能ファイルをセグメント 0x8–0x9 に挿入し、オプション 3 は実行可能ファイルをセグメント 0x9–0xA に挿入します。

非常に大容量のプログラム・アドレス・スペース・モデルの例

優先アドレス・スペース・モデルが次の場合:

- MAXDATA=0x50000000 DSA
- セグメント 0xB、0xC および 0xE に必要な shmat/mmap 領域
- 共用ライブラリー域テキストおよび事前再配置済みデータのアクセシビリティー

アドレス・スペースのレイアウトは、次のようになります。

```
0x0: System Segment
0x1:
0x2: Exec-time Private Dependencies / Stack
```

```
0x3: Process Heap
0x4:      |
0x5:      |
0x6:      v
0x7: ____________ (data limit)
0x8:
0x9:
0xA:
0xB: shmat/mmap (mapped after exec)
0xC: shmat/mmap (mapped after exec)
0xD: Shared Library Text
0xE: shmat/mmap (mapped after exec)
0xF: Pre-relocated Library Data
```

これから、セグメント 0x4–0xA が実行可能ファイル用に使用できることが分かります。

実行可能ファイルのサイズが 256 MB より大で、512 MB 未満であると仮定した場合、この状態に理想的な **HUGE_EXEC** 設定は次のとおりです。

1. HUGE_EXEC=0x8
2. HUGE_EXEC=0x9

オプション 1 は実行可能ファイルをセグメント 0x8–0x9 に挿入し、オプション 2 は実行可能ファイルをセグメント 0x9–0xA に挿入します。

注**:** (DSA のため) システムは実行可能ファイル用にセグメント 0xB–0xC を選択するので、このお客様には HUGE_EXEC=0 設定は適切ではありません。 これにより、実行可能ファイルの後で、shmat/mmap に対してこれらのセグメントを使用できなくなります。要求に応じて挿入を許可する一方で、**HUGE_EXEC** を 0x4、0x5、0x6、または 0x7 セグメントのいずれかに設定すると、プロセス・ヒープの成長は、要求された開始セグメントのすぐ下のセグメントに制限されます。

共用ライブラリー域にアクセスできない非常に大容量のプログラム・アドレス・スペース・モデルの例

ユーザーの優先アドレス・スペース・モデルが次の場合:

- MAXDATA=0xB0000000 DSA
- shmat/mmap 領域なし
- 共用ライブラリー域へのアクセシビリティーなし

アドレス・スペースのレイアウトは、次のようになります。

```
0x0: System Segment
0x1:
0x2: Exec-time Private Dependencies / Stack
0x3: Process Heap
0x4:      |
0x5:      |
0x6:      |
0x7:      |
0x8:      |
0x9:      |
0xA:      |
0xB:      |
0xC:      v
0xD: ____________ (data limit)
0xE:
0xF:
```

これから、セグメント 0x4–0xF が実行可能ファイル用に使用できることが分かります。

実行可能ファイルのサイズが 256 MB より大で、512 MB 未満であると仮定した場合、この状態に理想的な **HUGE_EXEC** 設定は次のとおりです。

1. HUGE_EXEC=0
2. HUGE_EXEC=0xE

両オプションとも実行可能ファイルをセグメント 0xE–0xF に挿入します。

注: 要求に応じて挿入を許可する一方で、HUGE_EXEC を 0x4-0xD セグメントのいずれかに設定すると、プロセス・ヒープの成長は、要求された開始セグメントのすぐ下のセグメントに制限されることになります。

デフォルトのプロセス・アドレス・スペース・モデルの例

優先アドレス・スペース・モデルが次の場合:

- MAXDATA=0 (非 DSA)
- shmat/mmap 領域なし
- 共用ライブラリー域テキストおよび事前再配置済みデータのアクセシビリティー

アドレス・スペースのレイアウトは、次のようになります。

```
0x0: System Segment
0x1:
0x2: Exec-time Private Dependencies / Process Heap / Stack
0x3:
0x4:
0x5:
0x6:
0x7:
0x8:
0x9:
0xA:
0xB:
0xC:
0xD: Shared Library Text
0xE:
0xF: Pre-relocated Library Data
```

これから、セグメント 0x3–0xC が実行可能ファイル用に使用できることが分かります。

実行可能ファイルのサイズが 256 MB より大で、512 MB 未満であると仮定した場合、この状態に理想的な **HUGE_EXEC** 設定は次のとおりです。

```
 1. HUGE_EXEC=0
 2. HUGE_EXEC=0x3
 ...
10. HUGE_EXEC=0xB
```

オプション 1 と 2 の結果は同じで、実行可能ファイルがセグメント 0x3–0x4 に挿入されます。

単一の共用ライブラリー域テキスト・セグメントのある **shtext_in_one** の例

優先アドレス・スペース・モデルが次の場合:

- MAXDATA=0x70000000 (非 DSA)
- セグメント 0xC、0xD、0xE および 0xF に必要な shmat/mmap 領域
- 共用ライブラリー域のアクセシビリティー

アドレス・スペースのレイアウトは、次のようになります。

```
0x0: System Segment
0x1: Shared Library Text
0x2: Exec-time Private Dependencies / Stack
0x3: Process Heap
0x4: Process Heap
0x5: Process Heap
0x6: Process Heap
0x7: Process Heap
0x8: Process Heap
0x9: Process Heap
0xA:
0xB:
0xC: shmat/mmap (mapped after exec)
0xD: shmat/mmap (mapped after exec)
0xE: shmat/mmap (mapped after exec)
0xF: shmat/mmap (mapped after exec)
```

これから、セグメント 0xA–0xB が実行可能ファイル用に使用できることが分かります。

実行可能ファイルのサイズが 256 MB より大で 512 MB 未満であると仮定した場合、この状態に理想的な **HUGE_EXEC** 設定は次のとおりです。

1. `HUGE_EXEC=0,shtext_in_one`
2. `HUGE_EXEC=0xA,shtext_in_one`

両オプションとも実行可能ファイルをセグメント 0xA–0xB に挿入し、共用ライブラリー・テキストをセグメント 0x1 に挿入します。

注**:** 要求に応じて挿入を許可する一方で、HUGE_EXEC を 0xB–0xE のいずれかに設定すると、実行可能ファイルの後、セグメント 0xC–0xF の一部が shmat/mmap 用に使用可能になりません。

2 つの共用ライブラリー域テキスト・セグメントのある **shtext_in_one** の例

優先アドレス・スペース・モデルが次の場合:

- `MAXDATA=0x70000000 DSA`
- セグメント 0xA および 0xB に必要な shmat/mmap 領域
- (**doubletext32** 属性により作成された) 共用ライブラリー域のテキストのアクセシビリティー

アドレス・スペースのレイアウトは、次のようになります。

```
0x0: System Segment
0x1: Shared Library Text (primary)
0x2: Exec-time Private Dependencies / Stack
0x3: Process Heap
0x4:      |
0x5:      |
0x6:      |
0x7:      |
0x8:      v
0x9: ___________ (data limit)
0xA: shmat/mmap (mapped after exec)
0xB: shmat/mmap (mapped after exec)
0xC:
0xD:
0xE:
0xF: Shared Library Text (secondary)
```

これから、セグメント 0xC–0xE が実行可能ファイル用に使用できることが分かります。

実行可能ファイルのサイズが 512 MB より大で 768 MB 未満であると仮定した場合、この状態に理想的な **HUGE_EXEC** 設定は次のとおりです。

1. `HUGE_EXEC=0,shtext_in_one`
2. `HUGE_EXEC=0xC,shtext_in_one`

両オプションとも実行可能ファイルをセグメント 0xC–0xE に挿入し、共用ライブラリー・テキストをセグメント 0x1 および 0xF に挿入します。

注: 要求に応じて挿入を許可する一方で、HUGE_EXEC を 0x4–0x7 のいずれかに設定すると、プロセス・ヒープの成長は、要求された開始セグメントのすぐ下のセグメントに制限されることになります。

ASO 環境変数

システム全体の Active System Optimizer (ASO) チューナブルは、**asoo** コマンドを使用して管理されます。 ASO 環境変数を使用して、単一プロセスの ASO 設定をカスタマイズすることができます。

ASO_ENABLED

項目	ディスクリプター
目的：	ASO 最適化にプロセスを明示的に組み込むか除外するのに使用されます。
値：	デフォルト: ASO 最適化基準を満たす場合、ASO はプロセスを最適化します。 可能な値: ALWAYS または NEVER ALWAYS - ASO はこのプロセスを優先させます。 NEVER - ASO はこのプロセスを最適化しません。
表示：	**echo $ASO_ENABLED** この値は内部でオンにされるので、初期デフォルト値は **echo** コマンドでは表示されません。
変更：	**ASO_ENABLED=[ALWAYS|NEVER] export ASO_ENABLED** 変更は、この変数が設定された後に実行されるプロセスに影響を与えます。 この変更は、ユーザーがこのシェルをログアウトするまで有効です。 永続的な変更を行うには、 **ASO_ENABLED=[ALWAYS|NEVER]** コマンドを `/etc/environment` ファイルに追加します。
診断：	N/A
チューニング：	N/A

カーネルのチューナブル・パラメーター

AIX カーネルのチューニング・パラメーターは、次の 6 グループに分類されます。すなわち、スケジューラーとメモリー・ロード制御のチューニング、VMM チューニング、同期入出力チューニング、非同期入出力チューニング、ディスクとディスク・アダプター・チューナブル、およびプロセス間通信チューナブルです。

変更

AIX では、ほとんどの AIX カーネルのチューニング・パラメーターを設定するための、柔軟な集中モードが提供されます。

今回のバージョンでは、**rc** ファイルの編集なしで、永続的な変更を行えるようになりました。この変更は、すべてのチューナブル・パラメーターのリブート値を、新しい `/etc/tunables/nextboot` スタンザ・ファイルの中に配置することで実現しました。マシンをリブートすると、このファイルの値が自動的に適用されます。

/etc/tunables/lastboot スタンザ・ファイルは自動的に生成され、この中にはリブート直後に設定されたすべての値が入れられます。 これにより、これらの値をいつでも返すことができます。/etc/tunables/lastboot.log ログ・ファイルは、リブート中のすべての変更と変更できなかった内容を記録します。

チューナブル・ファイルを変更するための、以下のコマンドが用意されています。

コマンド	目的
tunsave	スタンザ・ファイルに値を保存する
tunchange	スタンザ・ファイル内の値を更新する
tunrestore	ファイル内に指定されている該当パラメーター値を適用する
tuncheck	手動で作成されたファイルの妥当性検査を実行する
tundefault	チューナブル・パラメーターをデフォルト値にリセットする

これらのコマンドはすべて、現行のチューナブル・パラメーターと、リブート・チューナブル・パラメーターの両方の値に適用できます。 詳しくは、それぞれのマニュアル・ページを参照してください。

これらのカーネルのチューニング・パラメーターの変更について詳しくは、*Performance Tools Guide and Reference* の『Kernel Tuning』のセクションを参照してください。

vmtune および schedtune コマンドの置き換え

vmtune および **schedtune** コマンドは **vmo**、**ioo**、および **schedo** コマンドで置き換えられました。 **vmo** と **ioo** コマンドをまとめて **vmtune** の代わりに使用し、**schedo** コマンドを **schedtune** の代わりに使用します。 既存のパラメーターはすべて、新しいコマンドで使用されます。

ioo コマンドは、すべての入出力関連チューニング・パラメーターを管理し、 **vmo** コマンドは、以前 **vmtune** コマンドで管理されていたその他すべての仮想メモリー・マネージャー (VMM) のパラメーターを管理します。 これらの 3 つのコマンドは bos.perf.tune ファイルセットに含まれます。このファイルセットには、**tunsave**、**tunrestore**、**tuncheck**、および **tundefault** コマンドも含まれています。 bos.adt.samples ファイルセットには **vmtune** および **schedtune** コマンドが引き続き含まれます。これらは必要に応じて **vmo**、**ioo**、および **schedo** コマンドを呼び出す互換性シェル・スクリプトです。 これらの互換性スクリプトは、対話式に変更できるパラメーターへの変更のみをサポートします。つまり、**bosboot** を必要とし、さらにマシンをリブートしないと有効にならないパラメーターは、**vmtune** スクリプトでサポートされなくなった、ということです。ユーザーがこのようなパラメーターを変更する場合、**vmo -r** コマンドを使用しなければならなくなりました。該当する **vmtune** コマンドのオプションとパラメーターは、次のとおりです。

以前の **vmtune** オプション	使用法	新しいコマンド
-C 0|1	ページ・カラー化	**vmo -r -o pagecoloring=0|1**
-g *n1* -L *n2*	予約するラージ・ページのラージ・ページ・サイズ数	**vmo -r -o lgpg_size=n1 -o lgpg_regions=n2**
-v *n*	メモリー・プール当たりのフレーム数	**vmo -r -o framesets=n**
-i *n*	特殊データ・セグメント ID の間隔	**vmo -r -o spec_dataseg_int=n**
-V *n*	予約する特殊データ・セグメント ID の数	**vmo -r -o num_spec_dataseg=n**
-y 0|1	p690 メモリー親和性	**vmo -r -o memory_affinity=0|1**

vmtune と **schedtune** の互換性スクリプトは AIX の出荷には含まれていません。ご使用の設定を新規コマンドに移行するには、以下の表を参照してください。

schedtune オプション	**schedo** の同等オプション	機能
-a number	-o affinity_lim=number	SCHED_FIF02 ポリシーがスレッドを優先しなくなるまでのコンテキスト切り替え回数を設定する。
-b number	-o idle_migration_barrier=number	アイドル移行バリアを設定する。
-c number	-o %usDelta=number	クロック・ドリフトの調整を制御する。
-d number	-o sched_D=number	CPU 使用率を減衰させるのに使用されるファクターを設定する。
-e number	-o v_exempt_seconds=number	最後に中断して再開したプロセスが再中断するまでの時間を設定する。
-f number	-o pacefork=number	失敗した **fork** コールを再試行するまでのクロックの目盛り遅延数を設定する。
-F number	-o fixed_pri_global=number	グローバル実行キューに固定優先順位スレッドを保持する。
-h number	-o v_repage_hi=number	プロセス中断の始まりと終わりを判別するのに使用されるシステム全体の基準を変更する。
-m number	-o v_min_process=number	最小マルチプログラミング・レベルを設定する。
-p number	-o v_repage_proc=number	どのプロセスを中断するかを判別するのに使用されるプロセスごとの基準を変更する。
-r number	-o sched_R=number	CPU 使用率を集計する頻度を設定する。
-s number	-o maxspin=number	スリープ状態になるまでにロック上でスピンする回数を設定する。
-t number	-o timeslice=number	10 ms タイム・スライスの数を設定する。
-w number	-o v_sec_wait=number	スラッシングが終了してプロセスを混用に追加し直すまでの待機秒数を設定する。

vmtune オプション	**vmo** の同等オプション	**ioo** の同等オプション	機能
-b number		-o numfsbuf=number	ファイルシステム bufstructs の数を設定する。
-B number		-o hd_pbuf_cnt=number	このパラメーターは、*pv_min_pbuf* パラメーターに置換されました。
-c number		-o numclust=number	遅延書き込みによって処理される 16 KB クラスターの数を設定する。
-C 0|1	-r -o pagecoloring=0|1		特定のハードウェア・プラットフォームのページ・カラー化を使用可能または使用不可にする。
-d 0|1	-o deffps=0|1		据え置きページング・スペース割り振りをオンまたはオフにする。
-e 0|1		-o jfs_clread_enabled=0|1	JFS がすべてのファイルに対してクラスター読み取りを使用するかどうかを制御する。
-E 0|1		-o jfs_use_read_lock=0|1	ファイルからの読み取り時に JFS が共用ロックを使用するかどうかを制御する。
-f number	-o minfree=number		フリー・リストのフレームの数を設定する。
-F number	-o maxfree=number		フレーム・スチーリングを停止するフリー・リストのフレームの数を設定する。
-g number	-o lgpg_size number		ハードウェアでサポートされるラージ・ページのサイズを設定する (バイト単位)。

vmtune オプション	**vmo** の同等オプション	**ioo** の同等オプション	機能
-H number		-o pgahd_scale_thresh=number	システムが先読みの規模を縮小する mempool 内のフリー・ページ数を設定する。
-i number	-r -o spec_dataseg_int=number		特殊データ・セグメント ID を予約するときに使用する時間間隔を設定する。
-j number		-o j2_nPagesPerWriteBehindCluster=number	後書きクラスター当たりのページ数を設定する。
-J number		-o j2_maxRandomWrite=number	ランダム書き込みのしきい値カウントを設定する。
-k number	-o npskill=number		プロセスの kill を開始するページング・スペースのページ数を設定する。
-l number	-o lrubucket=number		最長未使用時間ページ置換バケット・サイズのサイズを設定する。
-L number	-o lgpg_regions=number		予約するラージ・ページ数を設定する。
-M number	-o maxpin=number		固定できる実メモリーの最大パーセントを指定する。
-n number	-o nokilluid=number		ページング・スペースが小さくても kill されないプロセスの **UID** の範囲を指定する。
-N number		-o pd_npages=number	ファイルを削除するときに RAM から 1 つのチャンク (大きい塊) で削除するページの数を指定する。
-p number	-o minperm%=number		ファイル・ページが repage アルゴリズムから保護される上限ポイントを設定する。
-P number	-o maxperm%=number		ページ・スチール・アルゴリズムがファイル・ページのみスチールする下限ポイントを設定する。
-q number		-o j2_minPageReadAhead=number	先読みするページの最小数を設定する。
-Q number		-o j2_maxPageReadAhead=number	先読みするページの最大数を設定する。
-r number		-o minpgahead=number	順次先読みを開始するページの数を設定する。
-R number		-o maxpgahead=number	先読みするページの最大数を設定する。
-s 0|1		-o sync_release_ilock=0|1	**sync** 中に i ノード・ロックを保持する時間を最小化するコードを使用可能または使用不可にする。
-S 0|1	-o v_pinshm=0|1		**shmget** システム・コール時に **SHM_PIN** フラグを使用可能または使用不可にする。
-t number	-o maxclient%=number		ページ・スチール・アルゴリズムがクライアントのファイル・ページのみスチールする下限ポイントを設定する。
-T number	-o pta_balance_threshold=number		新規 PTA セグメントが割り当てられるポイントを設定する。
-u number	-o lvm_bufcnt=number		ロー物理 I/O の LVM バッファー数を設定する。
-v number	-r -o framesets=number		メモリー・プール当たりのフレーム・セットの数を設定する。
-V number	-r -o num_spec_dataseg=number		予約する特殊データ・セグメント ID の数を設定する。

vmtune オプション	**vmo** の同等オプション	**ioo** の同等オプション	機能
-w number	-o npswarn=number		SIGDANGER シグナルがプロセスに送信されるフリー・ページング・スペース・ページの数を設定する。
-W number		-o maxrandwrt=number	遅延書き込みアルゴリズムを使用してページがディスクに同期するまでにランダム書き込みが RAM に累積するしきい値を設定する。
-y 0|1	-r -o memory_affinity=0|1		このパラメーターは存在しません。 メモリー親和性は常にオンです (ハードウェアがサポートしている場合)。
-z number		-o j2_nRandomCluster=number	ランダム書き込みのしきい値の距離を設定します。
-Z number		-o j2_nBufferPerPagerDevice=number	ページャー・デバイス当たりのバッファーの数を設定します。

no および nfso コマンドの拡張

no および **nfso** コマンドが拡張され、/etc/tunables/nextboot ファイルのチューナブル・パラメーターに対し、永続的な変更を行うことができるようになりました。 これらのコマンドには、任意のパラメーターに関するヘルプを表示する場合に使用する **-h** フラグがあります。

ヘルプ情報には以下の内容が含まれます。

- パラメーターの目的
- デフォルト、範囲、タイプなどの可能な値
- パラメーター値の変更時期決定のための診断とチューニング情報

チューニング・コマンド **ioo**、**nfso**、**no**、**vmo**、**raso**、**schedo** はすべて、共通の構文を使用します。 詳細な説明と、サポートされているチューニング・パラメーターの完全リストについては、それぞれのコマンドのマニュアル・ページを参照してください。

AIX 互換モード

互換モードに移行するときには **no** および **nfso** コマンドのみが適用されます。これは、**vmtune** および **schedtune** コマンドがなくなったためです。互換モードを使用して新しいチューニング・フレームワークに移行することができますが、AIX リリースでの使用はお勧めできません。

互換モードでは、ブート処理中に呼び出されるスクリプトに、チューニング・コマンドへの呼び出しを組み込むことによって、チューナブル・パラメーターへの変更を永続的にすることができます。相違点は、/etc/tunables/lastboot ファイルと /etc/tunables/lastboot.log ファイルがリブート中に作成されるということのみです。 lastboot.log ファイルには、AIX が現在互換モードで実行中であり、nextboot ファイルが適用されていないという警告が含まれます。

タイプ *Bosboot* のパラメーターを除き (482 ページの『vmtune および schedtune コマンドの置き換え』を参照)、チューニング・コマンドの新規のリブート・ オプションと永続オプション (それぞれ **-r** および **-p** フラグ) は、ファイルの内容がリブート時には適用されないため、意味がありません。 チューニング・コマンドは、非互換モードの場合とは異なり、パラメーターのリブート値を制御しません。 タイプ *Bosboot* のパラメーターは、移行中でも保持され、/etc/tunables/nextboot ファイルの中に保存されます。また、互換モードで実行されているかどうかにかかわりなく **-r** オプションを使用して変更できます。 /etc/tunables/nextboot ファイルを削除しないでください。

互換モードは、**pre520tune** という新規の **sys0** 属性によって制御されます。この属性は、移行インストール中に自動的に `enable` に設定されます。 この使用不可 (disable) モードでは、リブート中に呼び出されるスクリプトに組み込まれたチューニング・コマンドへの呼び出しが、`nextboot` ファイルの内容によって上書きされます。 **pre520tune** 属性の現在の設定は、次のコマンドを実行して表示できます。

```
# lsattr -E -l sys0
```

また、変更は、次のコマンドを使用するか、

```
# chdev -l sys0 -a pre520tune=disable
```

互換モードが使用不可の場合、**no** コマンド・パラメーターで、以下に示すリブート中にしか変更できないすべての *Reboot* タイプは、**-r** フラグを使用しないと変更できません。

- arptab_bsiz
- arptab_nb
- extendednetstats
- ifsize
- inet_stack_size
- ipqmaxlen
- nstrpush
- pseintrstack

現在のリブート設定を保持しながら非互換モードに切り替えるには、最初に **pre520tune** 属性を変更してから、次のコマンドを実行します。

```
# tunrestore -r -f lastboot
```

このコマンドは、`lastboot` ファイルの内容を `nextboot` ファイルにコピーします。 チューニング・モードの詳細については、 *Performance Tools Guide and Reference*の『Kernel tuning』のセクションを参照してください。

AIX システムのリカバリー手順

リブート後にマシンが不安定で、**pre520tune** 属性が `enable` に設定されている場合、ユーザーはリブート中に呼び出されたスクリプトから、問題になっているチューニング・コマンドへの呼び出しを削除する必要があります。

リブート中にどのパラメーターが設定されたか確認するには、`/etc/tunables/lastboot` ファイルを参照して、# DEFAULT VALUE とマークされていないパラメーターを検索します。 チューナブル・パラメーターの内容について詳しくは、ファイル参照の『Tunables File Format』のセクションを参照してください。

代わりにすべてのチューナブル・パラメーターをデフォルト値にリセットする場合には、以下のステップを実行してください。

1. `/etc/tunables/nextboot` ファイルを削除します。
2. **pre520tune** 属性を `disable` に設定します。
3. **bosboot** コマンドを実行します。
4. マシンをリブートします。

スケジューラーおよびメモリー・ロード制御のチューナブル・パラメーター

スケジューラーおよびメモリー・ロード制御に関連した幾つかのパラメーターがあります。

スケジューラーおよびメモリー・ロード制御のチューナブル・パラメーターのほとんどは、**schedo** マニュアル・ページで詳しく説明されます。 以下は、その他の関連パラメーターの一部です。

1. **maxuproc** パラメーターのチューニング:

項目	ディスクリプター
目的：	ユーザー ID 当たりのプロセスの最大数を指定します。
値：	デフォルト：40。範囲：1 から 131072
表示：	**lsattr -E -l sys0 -a maxuproc**
変更：	**chdev -l sys0 -a maxuproc=***NewValue* 変更は即時に有効になり、ブートしても保存されます。 値を小さくした場合には、システム・ブートの後にのみ有効になります。
診断：	ユーザーは追加のプロセスを fork できません。
チューニング：	これは、ユーザーが作成するプロセスが多すぎないようにする保護機能です。

2. **ncargs** パラメーターのチューニング:

項目	ディスクリプター
目的：	**exec()** サブルーチンを実行する場合は、許される最大サイズの ARG/ENV リストを (4 KB ブロック単位で) 指定してください。
値：	デフォルト: 256。範囲: 256 から 1024
表示：	**lsattr -E -l sys0 -a ncargs**
変更：	**chdev -l sys0 -a ncargs=***NewValue* 変更は即時に有効になり、ブートしても保存されます。
診断：	**exec()** システム・コールに渡した引数リストが長すぎるために、ユーザーは追加のプロセスを実行できません。デフォルト値が小さいと、一部のプログラムが「`arg list too long`」エラー・メッセージを出して失敗することがあります。その場合は、前述の **chdev** コマンドを使用して **ncargs** 値を増やしてから、プログラムを再実行してみてください。
チューニング：	これは、引数リストが長すぎることが原因で **exec()** サブルーチンが失敗することを防ぐためのメカニズムです。 **ncargs** を大きい値にチューニングすると、システムのメモリー・リソースの制約が増すことに注意してください。

仮想メモリー・マネージャーのチューナブル・パラメーター

vmo コマンドは、仮想メモリー・マネージャーのチューナブル・パラメーターを管理します。

詳しくは、**vmo** コマンドを参照してください。

同期入出力のチューナブル・パラメーター

同期入出力には幾つかのチューナブル・パラメーターが使用可能です。

同期入出力のチューナブル・パラメーターのほとんどは、**ioo** マニュアル・ページで詳しく説明されます。以下は、その他の関連パラメーターの一部です。

1. **maxbuf**

項目	ディスクリプター
目的：	ブロック入出力バッファー・キャッシュ内の (4 KB) ページ数。
値：	デフォルト：20。範囲：20 から 1000
表示：	**lsattr -E -l sys0 -a maxbuf**
変更：	**chdev -l sys0 -a maxbuf=***NewValue* 変更は直ちに有効になり、効果は永続的です。 **-T** フラグを使用すると、変更は即時であり、次のブートまで続きます。 **-P** フラグを使用すると、変更は次のブートまで据え置かれ、効果は永続的です。
診断：	**sar —b** コマンドが、 `%rcache` および `%wcache` が低い状態で `breads` または `bwrites` を示した場合、このパラメーターをチューニングすることができます。

項目	ディスクリプター
チューニング：	このパラメーターは、通常の入出力がブロック入出力バッファー・キャッシュを使用しない場合、システムにパフォーマンス上の効果をほとんど与えません。

参照**:** 『非同期ディスク入出力のチューニング』

2. **maxpout**

項目	ディスクリプター
目的：	ファイルへの保留入出力の最大数を指定します。
値：	デフォルト: 8193、範囲: 0 から n (n は 4 の倍数に 1 を足したもの)
表示：	**lsattr -E -l sys0 -a maxpout**
変更：	**chdev -l sys0 -a maxpout=***NewValue* 変更は直ちに有効になり、効果は永続的です。 **-T** フラグを使用すると、変更は即時であり、次のブートまで続きます。 **-P** フラグを使用すると、変更は次のブートまで据え置かれ、効果は永続的です。
診断：	大量の順次ディスク出力のあるプログラムを実行しているときにフォアグラウンド応答時間が長くなる場合は、ディスク入出力の積極的なペーシングが必要となることがあります。順次パフォーマンスが受け入れ難いほど低下した場合は、入出力のペーシングを減らすか使用不可にする必要があります。
チューニング：	フォアグラウンド・パフォーマンスが受け入れられない場合は、**maxpout** と **minpout** の両方の値を減らしてください。順次パフォーマンスが受け入れられないほど低下した場合は、1 つまたは両方の値を増やすか、両方とも 0 に設定して入出力ペーシングを使用不可にしてください。

3. **minpout**

項目	ディスクリプター
目的：	**maxpout** に到達したプログラムが、ファイルへの書き込みを再開できるポイントを指定します。
値：	デフォルト: 4096、範囲: 0 から n (n は 4 の倍数で、少なくとも **maxpout** より 4 を引いた値になります)
表示：	**lsattr -E -l sys0 -a minpout**
変更：	**chdev -l sys0 -a minpout=***NewValue* 変更は直ちに有効になり、効果は永続的です。 **-T** フラグを使用すると、変更は即時であり、次のブートまで続きます。 **-P** フラグを使用すると、変更は次のブートまで据え置かれ、効果は永続的です。
診断：	大量の順次ディスク出力のあるプログラムを実行しているときにフォアグラウンド応答時間が長くなる場合は、ディスク入出力の積極的なペーシングが必要となることがあります。順次パフォーマンスが受け入れ難いほど低下した場合は、入出力のペーシングを減らすか使用不可にする必要があります。
チューニング：	フォアグラウンド・パフォーマンスが受け入れられない場合は、**maxpout** と **minpout** の両方の値を減らしてください。順次パフォーマンスが受け入れられないほど低下した場合は、1 つまたは両方の値を増やすか、両方とも 0 に設定して入出力ペーシングを使用不可にしてください。

4. **mount -o nointegrity**

項目	ディスクリプター
目的：	新規の mount オプション (nointegrity) は、特定の書き込み集中型アプリケーションのためのローカル・ファイルシステムのパフォーマンスを改善することができます。 この最適化は、基本的に JFS ログへの書き込みを除去します。 このパフォーマンスの改善は、メタデータの整合性を犠牲にして達成されることに注意してください。 したがって、このオプションを指定してマウントしたファイルシステムは、システム・クラッシュによって回復不能になる可能性があるので、このオプションは特に注意して使用してください。 それにもかかわらず、特定のクラスのアプリケーションの場合には、システム・クラッシュの後のファイル・データの整合性が不要なので、nointegrity オプションの使用によって、犠牲を払わずにパフォーマンスを改善することができます。 nointegrity ファイルシステムの使用が有利である 2 つの例は、コンパイラーの一時ファイルの場合、およびノンマイグレーションの実行、すなわち mksysb インストールの場合です。

5. ページング・スペースのサイズ

項目	ディスクリプター
目的：	作業用ストレージのページを保持するために必要なディスク・スペースの量。
値：	デフォルト：構成に依存します。範囲：hd6 の場合は 32 MB から n MB、hd6 以外の場合は 16 MB から n MB
表示：	**lsps -a mkps** または **chps** または **smitty pgsp**
変更：	変更は直ちに有効になり、効果は永続的です。 ただし、ページング・スペースが直ちに使用されるとは限りません。
診断：	実行: **lsps -a**。プロセスがページング・スペースの不足が理由で削除された場合には、**psdanger()** サブルーチンを使用して状況をモニターしてください。
チューニング：	通常のワークロードを処理するための十分なページング・スペースがないと考えられる場合には、別の物理ボリューム上に新規のページング・スペースを追加するか、または既存のページング・スペースを大きくしてください。

6. **syncd Interval**

項目	ディスクリプター
目的：	**syncd** による **sync()** コール間の時間。
値：	デフォルト：60。範囲：1 から任意の正整数
表示：	**grep syncd** /sbin/rc.boot **vi** /sbin/rc.boot または
変更：	変更は次のブートから有効になり、効果は永続的です。 代替の方法では、**kill** コマンドを使用して **syncd** デーモンを終了し、コマンド /usr/sbin/syncd interval を使用してコマンド・ラインからそれを再始動します。
診断：	ファイルへの入出力が、**syncd** の実行中にブロックされます。
チューニング：	デフォルト・レベルでは、このパラメーターによるパフォーマンスの低下はほとんどありません。 変更はお勧めしません。データ保全性 (HACMP™ に関する) のために **syncd interval** を大幅に減少させると、パフォーマンスが低下する可能性があります。

非同期入出力のチューナブル値の変更

すべての AIO チューナブルには現行値、デフォルト値、最小値、および最大値があります。これらの値は、**ioo** コマンドを使用して表示することができます。

ioo コマンドを使用して変更できるのは現在値のみです。 他の 3 つの値は固定されており、チューナブルの境界をユーザーに知らせる目的で提示されているものです。 チューナブルの現在値はいつでも変更でき、オペレーティング・システムを再始動しても持続されるようにすることができます。 チューナブルはすべて、パフォーマンス・ツール・ファイルセットにある **ioo** コマンドによって制御される通常のルールおよびオプションに従います。

次の表は、制限のないチューナブルを要約したものです。

項目	説明
minservers	AIO 処理専用のプロセッサーごとのカーネル・プロセスの最小数を示します。 各カーネル・プロセスはメモリーを使用するので、予期される AIO の量が少ない場合、プロセッサーの数で乗算したときに、minservers チューナブル値が大きくならないようにする必要があります。 minservers チューナブルのデフォルト値は 3 です。
maxservers	AIO 処理専用のプロセッサーごとのカーネル・プロセスの最大数を示します。 このチューナブル値は、プロセッサーの数で乗算した場合、一度に進行中のスロー・パス入出力要求の制限を示し、起こりうる入出力の並行性の制限を表します。 maxservers チューナブルのデフォルト値は 30 です。
maxreqs	一度に未解決の状態にしておくことができる AIO 要求の最大数を示します。 要求には、進行中の要求のほか、開始されるのを待機している要求が含まれます。 AIO 要求の最大数は、/usr/include/sys/limits.h ファイルに定義されている AIO_MAX より小さくすることはできませんが、それより大きくすることはできます。 大量の AIO のあるシステムで、AIO 要求の最大数が AIO_MAX を超えるのは適切なことです。 maxreqs チューナブルのデフォルト値は 16384 です。

項目	説明
server_inactivity	サーバーが AIO 要求の処理を行わずにアイドル (スリープ) 状態の場合に、そのサーバーが終了されるまでのタイムアウト値 (秒単位) を示します。 終了によりサーバーの合計数が minservers * 数未満の CPU になると、サーバーは対応する AIO が処理を行うのを待機してスリープ状態に戻ります。 このメカニズムは、AIO 要求の処理に使用されないでいるスリープ中のプロセスの数を削減することで、全体的なシステム・パフォーマンスの改善に役立ちます。 server_inactivity チューナブルのデフォルト値は 300 です。

ディスクおよびディスク・アダプターのチューナブル・パラメーター

AIX オペレーティング・システムには、いくつかのディスクおよびディスク・アダプターのカーネル・チューナブル・パラメーターがあります。

1. ディスク・アダプターの未処理要求の制限

項目	説明
目的：	SCSI バス上で未処理でいられる要求の最大数。 (SCSI-2 Fast/Wide アダプターにのみ適用されます。)
値：	デフォルト：40。範囲：40 から 128
表示：	**lsattr -E -l scsi***n* **-a num_cmd_elems**
変更：	**chdev -l scsi***n* **-a num_cmd_elems=***NewValue*
	変更は直ちに有効になり、効果は永続的です。 **-T** フラグを使用すると、変更は即時であり、次のブートまで続きます。 **-P** フラグを使用すると、変更は次のブートまで据え置かれ、効果は永続的です。
診断：	ストライプされたロー論理ボリュームへの大量の書き込みを実行中のアプリケーションが、望ましいスループット率を取得していません。
チューニング：	値は、SCSI バス上の物理ドライブの数 (ディスク・アレイ内のドライブを含む) に、個々のドライブのキュー・エントリー数を乗じた値に等しくなければなりません。

2. ディスク・ドライブのキュー・エントリー数

項目	説明
目的：	ディスク装置がそのキューに保持できる要求の最大数。
値：	デフォルト: IBM disks=3、 IBM 以外 disks=0。範囲：製造会社によって指定されます。
表示：	**lsattr -E -l hdisk***n*
変更：	**chdev -l hdisk***n* **-a q_type=simple -a queue_depth=***NewValue*
	変更は直ちに有効になり、効果は永続的です。 **-T** フラグを使用すると、変更は即時であり、次のブートまで続きます。 **-P** フラグを使用すると、変更は次のブートまで据え置かれ、効果は永続的です。
診断：	N/A
チューニング：	IBM 以外のディスク・ドライブが要求をキューに入れられる場合には、オペレーティング・システムがこの能力を利用できるように、この変更を行ってください。

参照：『SCSI アダプターとディスク装置キューの制限の設定』

3. ファイバー・チャネル・アダプターの未処理要求の制限

項目	説明
目的：	ファイバー・チャネル・アダプターにおける未処理要求の最大数。
値：	デフォルト: 200。範囲: 200 から 4096
表示：	**lsattr -E -l fcs***n* **-a num_cmd_elems**

項目	説明
変更：	**chdev -l fcs***n* **-a num_cmd_elems=***NewValue*. この属性をすぐに変更するには、fcs*n* アダプターが defined 状態になっている必要があります。その他の場合、属性を変更するには **-P** フラグを使用します。 **-P** フラグは次のブート操作まで変更を据え置き、その変更は永続的なものになります。 **注:** デフォルト値と範囲の値は、ファイバー・チャネル・デバイスごとに異なります。一部の Fibre Channel および Fibre Channel over Ethernet (FC/FCoE) アダプターでは、指定できる **num_cmd_elems** パラメーターの最大値は、オブジェクト・データ・マネージャー (ODM) で意図された最大範囲より小さくなる可能性があります。 **chdev** コマンドで **num_cmd_elems** パラメーターに指定された値が、FC/FCoE アダプターによってサポートされている値より大きい場合、それらのアダプターについてエラー・メッセージが記録されます。
チューニング：	最適なパフォーマンスを得るためには、**num_cmd_elems** パラメーターの値を、サポートされる最大範囲に設定してください。

プロセス間通信のチューナブル・パラメーター

AIX には、プロセス間通信のチューナブル・パラメーターがあります。

1. **msgmax** パラメーターのチューニング:

項目	ディスクリプター
目的：	最大のメッセージ・サイズを指定します。
値：	最大値 4 MB で動的です。
表示：	N/A
変更：	N/A
診断：	N/A
チューニング：	カーネルにより必要に応じて動的に調整されるので、チューニングは不要です。

2. **msgmnb** パラメーターのチューニング:

項目	ディスクリプター
目的：	キュー上のバイトの最大数を指定します。
値：	最大値 4 MB で動的です。
表示：	N/A
変更：	N/A
診断：	N/A
チューニング：	カーネルにより必要に応じて動的に調整されるので、チューニングは不要です。

3. **msgmni** パラメーター:

項目	ディスクリプター
目的：	メッセージ・キュー ID の最大数を指定します。
値：	最大値 131072 で動的です。
表示：	N/A
変更：	N/A
診断：	N/A
チューニング：	カーネルにより必要に応じて動的に調整されるので、チューニングは不要です。

4. **msgmnm** パラメーター:

項目	ディスクリプター
目的：	キュー当たりのメッセージの最大数を指定します。
値：	最大値 524288 で動的です。
表示：	N/A
変更：	N/A
診断：	N/A
チューニング：	カーネルにより必要に応じて動的に調整されるので、チューニングは不要です。

5. **semaem** パラメーター:

項目	ディスクリプター
目的：	出口に関する調整のための最大数を指定します。
値：	最大値 16384 で動的です。
表示：	N/A
変更：	N/A
診断：	N/A
チューニング：	カーネルにより必要に応じて動的に調整されるので、チューニングは不要です。

6. **semmni** パラメーターのチューニング:

項目	ディスクリプター
目的：	セマフォー ID の最大数を指定します。
値：	最大値 131072 で動的です。
表示：	N/A
変更：	N/A
診断：	N/A
チューニング：	カーネルにより必要に応じて動的に調整されるので、チューニングは不要です。

7. **semmsl** パラメーターのチューニング:

項目	ディスクリプター
目的：	ID 当たりのセマフォーの最大数を指定します。
値：	最大値 65535 で動的です。
表示：	N/A
変更：	N/A
診断：	N/A
チューニング：	カーネルにより必要に応じて動的に調整されるので、チューニングは不要です。

8. **semopm** パラメーターのチューニング:

項目	ディスクリプター
目的：	**semop()** コール当たりのオペレーションの最大数を指定します。
値：	最大値 1024 で動的です。
表示：	N/A
変更：	N/A
診断：	N/A
チューニング：	カーネルにより必要に応じて動的に調整されるので、チューニングは不要です。

9. **semume** パラメーターのチューニング:

項目	ディスクリプター
目的：	プロセス当たりの undo エントリーの最大数を指定します。
値：	最大値 1024 で動的です。
表示：	N/A
変更：	N/A
診断：	N/A
チューニング：	カーネルにより必要に応じて動的に調整されるので、チューニングは不要です。

10. **semvmx** パラメーターのチューニング:

項目	ディスクリプター
目的：	セマフォーの最大値を指定します。
値：	最大値 32767 で動的です。
表示：	N/A
変更：	N/A
診断：	N/A
チューニング：	カーネルにより必要に応じて動的に調整されるので、チューニングは不要です。

11. **shmmax** パラメーターのチューニング:

項目	ディスクリプター
目的：	最大共有メモリー・セグメントのサイズを指定します。
値：	最大値が 32 ビット・プロセスの場合は 256 MB、64 ビット・プロセスの場合は 0x80000000u で、動的です。
表示：	N/A
変更：	N/A
診断：	N/A
チューニング：	カーネルにより必要に応じて動的に調整されるので、チューニングは不要です。

12. **shmmin** パラメーターのチューニング:

項目	ディスクリプター
目的：	最小共有メモリー・セグメントのサイズを指定します。
値：	最小値 1 で動的です。
表示：	N/A
変更：	N/A
診断：	N/A
チューニング：	カーネルにより必要に応じて動的に調整されるので、チューニングは不要です。

13. **shmmni** パラメーターのチューニング:

項目	ディスクリプター
目的：	共有メモリー ID の最大数を指定します。
値：	最大値 1048576 で動的です。
表示：	N/A
変更：	N/A
診断：	N/A
チューニング：	カーネルにより必要に応じて動的に調整されるので、チューニングは不要です。

ネットワークのチューナブル・パラメーター

ネットワークのチューナブル・パラメーターには、ネットワーク・オプションと NFS オプションの 2 つのグループがあります。

ネットワーク・オプションのチューナブル・パラメーター

AIX には、ネットワーク・オプションのチューナブル・パラメーターに関連した幾つかのパラメーターがあります。

ネットワーク・オプションのチューナブル・パラメーターのほとんどは、*Commands Reference, Volume 4*の **no** マニュアル・ページで詳しく説明されます。 SP 環境で特別な注意が必要なネットワークのチューナブル・パラメーターについて詳しくは、「*RS/6000 SP System Performance Tuning*」を参照してください。
以下は、その他の関連パラメーターの一部です。

1. **maxmbuf**

項目	ディスクリプター
目的：	MBUFS に許容される実メモリーの最大キロバイト
値：	デフォルト：0。範囲：x から y
表示：	**lsattr -E -l sys0 -a maxmbuf**
変更：	**chdev -l sys0 -a maxmbuf**=*NewValue* 変更は直ちに有効になり、効果は永続的です。 **-T** フラグを使用すると、変更は即時であり、次のブートまで続きます。 **-P** フラグを使用すると、変更は次のブートまで据え置かれ、効果は永続的です。
診断：	N/A
チューニング：	*maxmbuf* が 0 より大きい場合は、 *thewall* の値に関係なく *maxmbuf* 値が使用されます。 mbufs の上限は *maxmbuf* または *thewall* のどちらか大きい値になります。
参照：	318 ページの『mbuf プールをモニターするための netstat -m コマンド』

2. **MTU**

項目	ディスクリプター
目的：	ネットワーク上で送信されるパケットのサイズを制限します。
値：	デフォルト：構成に依存します。
表示：	**lsattr -E -l** *interface_name*
変更：	**chdev -l** *interface_name* **-a mtu**=*NewValue* **chdev** コマンドを使用する場合、インターフェースは使用中に変更できません。 変更はリブート間で有効です。 次の代替方式もあります。 **ifconfig** *interface_name* **mtu** *NewValue*。これは、実行中のシステムで MTU サイズを変更しますが、システムのリブート間でこの値は保存されません。
診断：	パケットのフラグメント化の統計情報。
チューニング：	ネットワーク・インターフェースの MTU サイズを増加してください。 ギガビット・イーサネット・アダプターの場合には、デバイス属性 **jumbo_frames=yes** を使用して、jumbo フレームを使用可能にしてください (インターフェースに関する MTU を 9000 に設定するだけでは不十分です)。
参照：	279 ページの『TCP および UDP のパフォーマンスのチューニング』

3. **rfc1323**

項目	ディスクリプター
目的：	RFC 1323 によって指定される TCP 拡張機能を使用可能にします (ハイパフォーマンスのための TCP 拡張機能)。 値 1 は、*tcp_sendspace* と *tcp_recvspace* が 64 KB を超えられることを示します。
値：	デフォルト：0。範囲：0 から 1
表示：	**lsattr -El** *interface* または **ifconfig** *interface*
変更：	**ifconfig** *interface* **rfc1323** *NewValue*OR **chdev -l** *interface* **-a rfc1323=***NewValue*
	ifconfig コマンドは値を一時的に設定して、テストに役立てます。 **chdev** コマンドは、ODM を変更するので、システムのリブート後にカスタム値に戻ります。
診断：	N/A
チューニング：	デフォルト値 0 は、システム全体の規模で RFC 拡張機能を使用不可にします。 値 1 は、すべての TCP 接続が RFC 拡張機能のネゴシエーションを試みることを指定します。 SOCKETS アプリケーションは、**setsockopt()** サブルーチンを使用して、個々の TCP 接続に関するデフォルトの動作をオーバーライドすることができます。これは実行時属性です。 *tcp_sendspace* および *tcp_recvspace* を、 64 KB を超える値に設定する前に、変更を行ってください。
参照：	294 ページの『TCP ワークロードのチューニング』

4. tcp_mssdflt

項目	ディスクリプター
目的:	リモート・ネットワークとの通信で使用されるデフォルトの最大セグメント・サイズ。
値：	デフォルト：512 バイト
表示：	**lsattr -El** *interface* または **ifconfig** *interface*
変更：	**ifconfig** *interface* **tcp_mssdflt** *NewValue*OR **chdev -l** *interface* **-a tcp_mssdflt=***NewValue*
	ifconfig コマンドは値を一時的に設定して、テストに役立てます。 **chdev** コマンドは、ODM を変更するので、システムのリブート後にカスタム値に戻ります。
診断：	N/A
チューニング：	*tcp_mssdflt* が使用されるのは、パス MTU ディスカバリーが使用可能でない場合、またはパス MTU ディスカバリーがパス MTU の発見に失敗した場合です。データを (MTU - 52) バイトに制限すると、可能な限り、フル・パケットのみが送信されます。 これは実行時属性です。
参照：	312 ページの『TCP 最大セグメント・サイズのチューニング』

5. tcp_nodelay

項目	ディスクリプター
目的：	このインターフェースで TCP を使用するソケットが、データの送信時に Nagle アルゴリズムに従うことを指定します。 デフォルトで、TCP は Nagle アルゴリズムに従います。
値：	デフォルト：0。範囲：0 または 1
表示：	**lsattr -El** *interface* または **ifconfig** *interface*
変更：	**ifconfig** *interface* **tcp_nodelay** *NewValue*OR **chdev -l** *interface* **-a tcp_nodelay=***NewValue*
	ifconfig コマンドは値を一時的に設定して、テストに役立てます。 **chdev** コマンドは、ODM を変更するので、システムのリブート後にカスタム値に戻ります。
診断：	N/A
チューニング：	これはインターフェース固有のネットワーク・オプション (ISNO) のオプションです。
参照：	290 ページの『インターフェース固有のネットワーク・オプション』

6. tcp_recvspace

項目　ディスクリプター
目的：　データを受信するためのシステム・デフォルトのソケット・バッファー・サイズを指定します。 これは、TCP が使用するウィンドウ・サイズに影響を与えます。
値：　デフォルト：16384 バイト
表示：　**lsattr -El** *interface* または **ifconfig** *interface*
変更：　**ifconfig** *interface* **tcp_recvspace** *NewValue*OR **chdev -l** *interface* **-a tcp_recvspace=***NewValue*

ifconfig コマンドは値を一時的に設定して、テストに役立てます。 **chdev** コマンドは、ODM を変更するので、システムのリブート後にカスタム値に戻ります。
診断：　N/A
チューニング：　ソケット・バッファー・サイズを 16 KB (16 384) に設定すると、標準のイーサネットおよびトークンリング・ネットワークでのパフォーマンスを改善することができます。 デフォルト値は 16 384 です。 シリアル・ライン・インターネット・プロトコル (SLIP) のような、より低い帯域幅のネットワーク、あるいは、シリアル・オプティカル・リンクのような、より高い帯域幅のネットワークの場合には、別の最適なバッファー・サイズが必要になります。 最適なバッファー・サイズは、メディアの帯域幅とパケットの平均往復時間の積です。

tcp_recvspace 属性は、 **sb_max** 属性の設定値以下のソケット・バッファー・サイズを指定する必要があります。 これは動的属性ですが、**inetd** デーモンによって始動されるデーモンについては、以下のコマンドを実行してください。

- **stopsrc-s inetd**
- **startsrc -s inetd**

参照：　294 ページの『TCP ワークロードのチューニング』

7. tcp_sendspace

項目　ディスクリプター
目的：　データを送信するためのシステム・デフォルトのソケット・バッファー・サイズを指定します。
値：　デフォルト：16384 バイト
表示：　**lsattr -El** *interface* または **ifconfig** *interface*
変更：　**ifconfig** *interface* **tcp_sendspace** *NewValue*OR **chdev -l** *interface* **-a tcp_sendspace=***NewValue*

ifconfig コマンドは値を一時的に設定して、テストに役立てます。 **chdev** コマンドは、ODM を変更するので、システムのリブート後にカスタム値に戻ります。
診断：　N/A
チューニング：　これは、TCP が使用するウィンドウ・サイズに影響を与えます。 ソケット・バッファー・サイズを 16 KB (16 384) に設定すると、標準のイーサネットおよびトークンリング・ネットワークでのパフォーマンスを改善することができます。デフォルト値は 16 384 です。 シリアル・ライン・インターネット・プロトコル (SLIP) のような、より低い帯域幅のネットワーク、あるいは、シリアル・オプティカル・リンクのような、より高い帯域幅のネットワークの場合には、別の最適なバッファー・サイズが必要になります。最適なバッファー・サイズは、メディアの帯域幅とパケットの平均往復時間の積です。

```
optimum_window=bandwidth * average_round_trip_time
```

tcp_sendspace 属性は、**sb_max** 属性の設定値以下のソケット・バッファー・サイズを指定する必要があります。 **tcp_sendspace** パラメーターは動的属性ですが、**inetd** デーモンによって始動されるデーモンについては、以下のコマンドを実行してください。

- **stopsrc-s inetd**
- **startsrc -s inetd**

参照：　294 ページの『TCP ワークロードのチューニング』

8. use_sndbufpool

項目	ディスクリプター
目的：	ソケットに送信バッファー・プールを使用するかどうかを指定します。
値：	デフォルト：1
表示：	**netstat -m**
変更：	このオプションは、値を 1 に設定すると使用可能になり、0 に設定すると使用不可になります。
診断：	N/A
チューニング：	これは、ロード時のブール・オプションです。

9. **xmt_que_size**

項目	ディスクリプター
目的：	インターフェースに関して、キューに入れることができる送信バッファーの最大数を指定します。
値：	デフォルト：構成に依存します。
表示：	**lsattr -E -l** *interface_name*
変更：	**ifconfig** *interface_name* **detach** chdev **-l** *interface_name* **-a***que_size_name***=***NewValue* **ifconfig** *interface_name hostname* **up**. インターフェースが使用中は、変更できません。 変更はリブート間で有効です。
診断：	**netstat -i** (0err > 0)
チューニング：	サイズを増加してください。
参照：	325 ページの『netstat コマンド』

NFS オプションのチューナブル・パラメーター

AIX には、NFS オプションのチューナブル・パラメーターに関連した幾つかのパラメーターがあります。

NFS オプションのチューナブル・パラメーターのほとんどは、**nfso** マニュアル・ページで詳しく説明されます。 以下は、その他の関連パラメーターの一部です。

1. **biod** カウント

項目	ディスクリプター
目的：	クライアント上で NFS 要求を処理するために使用可能な biod プロセスの数。
値：	デフォルト：6。範囲：1 から任意の正整数
表示：	**ps -efa | grep biod**
変更：	**chnfs -b** *NewValue* 変更は通常、直ちに有効になり効果は永続的です。 **-N** フラグによって、即時の、一時的な変更が行われます。 **-I** フラグによって、変更は次のブート時に有効になります。
診断：	**netstat -s** で UDP ソケット・バッファー・オーバーフローを調べます。
チューニング：	ソケット・バッファー・オーバーフローがなくなるまで数を増やします。
参照：	376 ページの『必要な biod スレッドの数』

2. **combehind**

項目	ディスクリプター
目的：	NFS バージョン 3 のマウントで非常に大きいファイルに書き込むときに、NFS クライアントで commit-behind 動作を使用可能にします。
値：	デフォルト：0。範囲：0 から 1
表示：	マウント
変更：	**mount -o combehind**
診断：	NFS バージョン 3 のマウントで非常に大きいファイル (基本的に NFS クライアントのシステム・メモリー量より大きいファイル) に書き込むときの、スループットの低下。
チューニング：	NFS の主な用途が NFS サーバーに非常に大きなファイルを書き込むことである場合、NFS クライアントでこのマウント・オプションを使用します。 このオプションの好ましくないフィーチャーは、NFS ファイルでの VMM キャッシングが、クライアントで使用不可になることです。 したがって、良好な NFS 読み取りパフォーマンスが必要な環境では、このオプションを使用しないことをお勧めします。

3. **nfsd** カウント

項目	ディスクリプター
目的：	着信 NFS 要求にサービスを提供するために作成される NFS サーバー・スレッドの最大数を指定します。
値：	デフォルト：3891。範囲：1 から 3891
表示：	**ps -efa | grep nfsd**
変更：	**chnfs -n** *NewValue* 変更は通常、直ちに有効になり効果は永続的です。 **-N** フラグによって、即時の、一時的な変更が行われます。 **-I** フラグによって、変更は次のブート時に有効になります。
診断：	**nfs_max_threads** を参照。
チューニング：	**nfs_max_threads** を参照。
参照：	376 ページの『必要な biod スレッドの数』

4. **numclust**

項目	ディスクリプター
目的：	**combehind** オプションと併用すると、NFS バージョン 3 のマウントで大きいファイルに書き込むときの、書き込みスループット・パフォーマンスが改善されます。
値：	デフォルト: 128。範囲: 8 から 1024
表示：	マウント
変更：	`mount -o numclust=`*`NewValue`*
診断：	NFS バージョン 3 のマウントで非常に大きいファイル (基本的に NFS クライアントのシステム・メモリー量より大きいファイル) に書き込むときの、スループットの低下。
チューニング：	NFS の主な用途が NFS サーバーに非常に大きなファイルを書き込むことである場合、NFS クライアントでこのマウント・オプションを使用します。 この値は、基本的に、VMM が NFS クライアントからコミット操作を生成しようとする対象のページの最小数を表します。 値が小さすぎると、コミット回数 (各コミットでサーバーへ同期書き込みが行われます) が多すぎるために、スループットが低下します。 値が大きすぎても、NFS クライアント・メモリーが変更されたページでいっぱいになり、それによって LRU デーモンがページの再利用を開始するために呼び出されるため、スループットが低下します。 **lrud** が実行されると、V3 の書き込みは基本的に同期となります。各書き込みがコミットと共に終了するためです。 この状態は、**numclust** オプションと **combehind** オプションを使用して回避できます。

ストリームのチューニング属性

ストリームのチューニング属性の詳細リストは、**-L** オプションを指定して **no** コマンドを実行すると表示されます。

テスト・ケースのシナリオ

各ケースについて、システムのタイプと起きた問題点が述べてあります。 その後、特定のパフォーマンス問題をテストする方法、および、問題が検出された場合の解決方法も説明しています。 ご使用の環境が似ている場合は、以下のテスト・ケースの情報が役に立ちます。

パフォーマンス・チューニングは、システムおよびアプリケーションに依存する部分が大ですが、ほとんどの AIX システムで効果のある汎用のチューニング方法がいくつかあります。

NFS クライアントにおける大容量ファイルの書き込みパフォーマンスの向上

NFS マウントされたファイルシステムに大きい順次ファイルを書き込むと、NFS サーバーへのファイル転送速度が大幅に低下する可能性があります。 このシナリオでは、この状況が起きているかどうかを特定し、この問題を修正する手順を使用します。

考慮事項

ここで解説する情報は AIX の特定バージョンを使用してテストされたものです。したがって、その内容は使用される AIX のバージョンおよびレベルによってかなり異なることがあります。

NFS マウントされたファイルシステムに (マシンの物理メモリー量を超える) 大容量ファイルを順次に書き込むアプリケーションを、システムが実行していると仮定します。 このファイルシステムは NFS V3 を使用してマウントされています。 NFS サーバーとクライアントは、100 MB/秒のイーサネット・ネットワーク上で通信します。 小さいファイルを順次に書き込む場合、スループットの平均は約 10 MB/秒です。 しかし、非常に大きいファイルを書き込む場合、スループットの平均は 1 MB/秒を大きく下回ります。

このアプリケーションの大容量ファイルの書き込みによりクライアントのメモリーが満杯になり、NFS サーバーへの転送速度が低下します。 この原因は、クライアント AIX システムがメモリー内の一部のページを解放して、アプリケーションで書き込まれる次のページのセットを受け入れるために LRUD **kproc** を呼び出すことにあります。

この問題が発生しているかどうかを判別するには、以下のいずれかの方法を使用します。

- ファイルが NFS サーバーに書き込まれている間に、次のように入力して **nfsstat** コマンドを一定期間 (10 秒) ごとに実行します。

  ```
  nfsstat
  ```

  nfsstat コマンド出力を確認します。 V3 コミット呼び出しの数が V3 書き込み呼び出しの数に対してほぼ直線的に増加している場合は、問題が発生している可能性が高いです。
- (`bos.perf.tools` ファイルセット内にある) **topas** コマンドを使用して次のように入力し、NFS サーバーに送信されている 1 秒当たりのデータ量をモニターします。

  ```
  topas -i 1
  ```

リストされているどちらの方法を使用しても問題が存在することが示される場合、その解決策は、クライアント・システムで NFS サーバー・ファイルシステムをマウントするときに、**combehind** という新しい **mount** コマンド・オプションを使用することです。 次の操作を実行します。

1. ファイルシステムがアクティブでないときに、次のように入力してこのファイルシステムをアンマウントします。

   ```
   unmount /mnt
   ```

   (/mnt がローカル・マウント・ポイントであることを前提とします)
2. **comebehind** という **mount** コマンド・オプションを次のように使用して、リモート・ファイルシステムを再マウントします。

   ```
   mount -o combehind server_hostname:/remote_mount_point /mnt
   ```

関連概念:

356 ページの『NFS パフォーマンス』
AIX は、サーバーとクライアントの両方で、ネットワーク・ファイルシステム (NFS) のモニターとチューニングを行うためのツールとメソッドを提供しています。

関連情報:

mount コマンド

nfsstat コマンド

topas コマンド

パスワードの索引付けを使用したセキュリティー・サブルーチンの簡素化

このシナリオでは、多数のセキュリティー・サブルーチン・プロセスがあることを確認し、パスワード・ファイルの索引付けを行うことにより、セキュリティー・サブルーチンのために使用されるプロセッサー時間を削減します。

考慮事項

ここで解説する情報は AIX の特定バージョンを使用してテストされたものです。したがって、その内容は使用される AIX のバージョンおよびレベルによってかなり異なることがあります。

このシナリオ環境では、1 つの 2-way システムがメール・サーバーとして使用されます。 メールは POP3 (Post Office Protocol Version 3) を介してリモートで受信されるか、サーバーに直接ログインできるローカル・メール・クライアントにより受信されます。 メールは **sendmail** デーモンを使用して送信されます。 メール・サーバーの特性により、ユーザー認証に対して多数のセキュリティー・サブルーチンが呼び出されます。 ユニプロセッサーのマシンから 2-way システムへの移動後、**uptime** コマンドは 200 プロセスを返しています。これに対して、ユニプロセッサーのマシンでは 1 より少ないプロセスです。

性能低下の原因を特定し、セキュリティー・サブルーチンのために使用されるプロセッサー時間を削減するには、次の操作を実行します。

1. プロセッサー時間を消費する割合が高いプロセスを特定し、プロセッサー時間の大半をカーネル・モードとユーザー・モードのどちらで費やしているかを判別するために、次のコマンド (`bos.perf.tools` ファイルセットに格納) を実行します。

   ```
   topas -i 1
   ```

   このシナリオの **topas** コマンド出力では、プロセッサー時間の大半 (約 90%) をユーザー・モードで費やし、ほとんどのプロセッサー時間を消費するプロセスが **sendmail** と **pop3d** であることが示されています。 (プロセッサーの大半をカーネル・モードで費やしている場合は、カーネル・トレースのツールを使用するのが適切です。)
2. ユーザー・モードのプロセッサー時間をアプリケーション・コード (ユーザー) または共有ライブラリー (共有) のどちらで費やしているかを判別するために、次のコマンドを実行して 60 秒間データを収集します。

   ```
   tprof -ske -x "sleep 60"
   ```

   この **tprof** コマンドにより、共有ライブラリーから呼び出されたサブルーチンの名前がリストされ、各サブルーチンに費やしたプロセッサーのティック数順にソートされます。 この場合、**tprof** データは、ユーザー・モードのプロセッサー時間の大半をセキュリティー・サブルーチン (およびそれに呼び出されるサブルーチン) に対する `libc.a` システム・ライブラリーで費やしていることを示していま

す。 (**tprof** コマンドが、ユーザー・モードのプロセッサー時間の大半をアプリケーション・コード (ユーザー) で費やしていることを示した場合は、アプリケーションのデバッグとプロファイルが必要になります。)

3. /etc/passwd ファイルを各セキュリティー・サブルーチンごとにスキャンしないようにするために、次のコマンドを実行してこのファイルの索引を作成します。

   ```
   mkpasswd -f
   ```

索引付けされたパスワード・ファイルを使用することで、このシナリオの負荷平均は 200 から 0.6 に減少しています。

詳細情報

- 「*Commands Reference, Volume 5*」の **topas**、**tprof**、および **uptime** コマンドの説明。
- 「*Commands Reference, Volume 4*」の **pop3d** および **sendmail** デーモンの説明。

BSR 共有メモリー

バリア同期レジスター (BSR) は、小規模または高密度のメモリー領域を効率的に共用するためのハードウェア機能です。 メモリー領域は複数のスレッドにより並行して更新されます。

BSR メモリーにより、格納されたメモリーがシステム内を通常のキャッシュされたメモリーよりも速く伝搬することができます。 BSR メモリーは、高度に並列化されたアプリケーションのために、複数のプロセッサーから少量のメモリーを操作するためのキャッシュ管理セマンティクスを使用します。 キャッシュ管理セマンティクスは、汎用の共有メモリーに使用するものではありません。 この機能は、ワークロードを高度に並列化した状況で使用されるバリア同期構造を効率的に実装するために役立ちます。

BSR メモリーには特別なプロセッサー・サポートが必要であり、論理区画 (LPAR) が BSR 機能を使用できるように、リソースを LPAR に構成する必要があります。

BSR 共有メモリーを割り当てるには、以下の手順を実行します。

1. shmctl() サブルーチンを使用して、BSR 共有メモリーのシステム V 共有メモリー領域を割り当てます。
2. shmctl() サブルーチンを使用し、**SHM_BSR** コマンドを指定することによって、割り振られたシステム V 共有メモリー領域が BSR メモリーによってバックアップされることを要求します。

   注**: SHM_BSR** コマンドと一緒に使用される shmctl() サブルーチンが、1 つの共有メモリーで使用されます。 このステップは、shmget() サブルーチンを使用してシステム V 共有メモリーが作成された直後、かつ任意のプロセスが共有メモリーに接続される前に実行されます。 **SHM_BSR** コマンドが使用されると、shmctl() サブルーチンは指定された共有メモリー領域に対して BSR メモリーを使用しようとします。
3. 使用可能な BSR メモリーが不足している場合、またはハードウェア・プラットフォームで BSR 機能がサポートされていない場合にエラーを表示します。 shmget() サブルーチンは失敗し、errno が ENOMEM に設定されます。

   注**:** root ユーザー以外のユーザーが BSR メモリーを割り当てるには、CAP_BYPASS_RAC_VMM 権限が必要です。 root ユーザー以外のユーザーがこの能力を持っていない場合は、**SHM_BSR** コマンドを設定した shmctl() サブルーチンは失敗し、errno が EPERM に設定されます。

BSR 共有メモリーを使用する場合、共有メモリー領域には 1 バイトおよび 2 バイトのストア命令のみ使用できます。 2 バイト超を使用するストア命令は、BSR 共有メモリーでは正しく機能しません。ロード命令では任意のサイズが BSR 共有メモリーにロードできます。

VMINFO コマンドを vmgetinfo() サブルーチンで使用して実行すると、使用可能な BSR サポートに関する情報を収集できます。 **VMINFO** コマンドが vmgetinfo() サブルーチンに指定されている場合は、vminfo 構造体が戻されます。 bsr_mem_total フィールドには、LPAR に構成された BSR メモリーの合計量が報告されます。 bsr_mem_free フィールドには、現在割り当て可能な BSR の合計量が報告されます。

shmctl() サブルーチンに SHM_SIZE オプションを指定しても、BSR 共有メモリー領域を動的にサイズ変更することはできません。 アプリケーションが shmctl() サブルーチンに SHM_SIZE パラメーターを指定して BSR 共有メモリー領域をサイズ変更しようとすると、shmctl() は失敗して errno が EINVAL に設定されます。 BSR 共有メモリーは、EXTSHM 環境変数がある場合は、サポートされません。 EXTSHM 環境変数が設定されている場合、SHM_BSR フラグを使用して shmctl() が呼び出されると、shmctl() は EINVAL で失敗します。

例

次の例は、アプリケーションがシステムで使用可能な BSR メモリーの量を照会して、BSR 共有メモリー領域を割り当て、接続できることを示しています。 この例は、アプリケーションがどのように BSR 共有メモリー領域を分離して削除するかを示しています。

```
#include <errno.h>
#include <stdio.h>
#include <sys/shm.h>
#include <sys/vminfo.h>

/* shm_rgn_size is the size of the shared memory region to
 * allocate. In this example, 4KB (PAGESIZE) is chosen.  4KB is the
 * smallest shared memory region size supported.  It is expected that
 * 4KB should be sufficient for most users of BSR memory.
 */
const size_t shm_rgn_size = PAGESIZE;

int main(int argc, char *argv[])
{
 struct vminfo my_info = { 0 };
 int           id;
 void          *ptr;

 /* Determine the amount of BSR memory available */
 if (vmgetinfo(&my_info, VMINFO, sizeof(my_info)) != 0)
 {
  perror("vmgetinfo() unexpectedly failed");
  return 1;
 }

 /* Check to see that sufficient BSR memory is available */
 if (my_info.bsr_mem_free < shm_rgn_size)
 {
  fprintf(stderr, "insufficient BSR memory¥n");
  return 2;
 }

 /* Allocate a new shared memory region */
 id = shmget(IPC_PRIVATE, shm_rgn_size, IPC_CREAT|IPC_EXCL);

 if (id == -1)
 {
  perror("shmget() failed");
  return 3;
```

```
 }

 /* Request BSR memory for the shared memory region */
 if (shmctl(id, SHM_BSR, NULL))
 {
  perror("shmctl(SHM_BSR) failed");
  shmctl(id, IPC_RMID, 0);
  return 4;
 }

 /* Attach the shared memory region */
 ptr = shmat(id, NULL, 0);
 if ((int)ptr == -1)
 {
  perror("shmat() failed");
  shmctl(id, IPC_RMID, 0);
  return 5;
 }

 /* BSR memory can now be accessed starting at address - ptr */

 /* Detach shared memory region */
 if (shmdt(ptr))
 {
  perror("shmdt() failed");
  shmctl(id, IPC_RMID, 0);
  return 6;
 }

 /* Delete shared memory region */
 if (shmctl(id, IPC_RMID, 0))
 {
  perror("shmctl(IPC_RMID) failed");
  return 7;
 }

 return 0;
}
```

VMM fork ポリシー

あるプロセスが fork される場合の各プロセス専用のメモリーの管理方法を変更することができます。

仮想メモリー・マネージャー (VMM) は、あるプロセスが fork されるときにそのプロセスのアドレス・スペース全体はコピーしません。ページは、親プロセスまたは子プロセスのいずれかによって変更されるときに、要求に応じてコピーされます。まだ変更されていないページへのロード参照は、親プロセスと子プロセスとの間で共有されるメモリーに解決されます。そのページが続けて変更される場合は、変更時にコピーされます。

メモリーがプロセスによって読み取られ、ただちに書き込まれる場合は、最初に書き込まれる時点ではなく、最初に参照される時点でページのコピーを作成する方が簡単です。この動作は、**vmo** コマンドを使用して制限付き **vmm_fork_policy** チューナブル・パラメーターを変更することにより、システム全体に使用することができます。 VMM_CNTRL 環境変数をエクスポートし、**vmm_fork_policy** キーワードを指定することで、単一プロセスを用いたグローバル・チューナブル・パラメーターをオーバーライドすることができます。

特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒*103-8510*
東京都中央区日本橋箱崎町*19*番*21*号
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

記載されている性能データとお客様事例は、例として示す目的でのみ提供されています。実際の結果は特定の構成や稼働条件によって異なります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。 IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向または意向に関する記述は、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

表示されている IBM の価格は IBM が小売り価格として提示しているもので、現行価格であり、通知なしに変更されるものです。卸価格は、異なる場合があります。

本書はプランニング目的としてのみ記述されています。記述内容は製品が使用可能になる前に変更になる場合があります。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、類似する個人や企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。これらのサンプル・プログラムは特定物として現存するままの状態で提供されるものであり、いかなる保証も提供されません。 IBM は、お客様の当該サンプル・プログラムの使用から生ずるいかなる損害に対しても一切の責任を負いません。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生した創作物には、次のように、著作権表示を入れていただく必要があります。

© (お客様の会社名) (西暦年).

このコードの一部は、IBM Corp. のサンプル・プログラムから取られています。

© Copyright IBM Corp. _年を入れる_.

プライバシー・ポリシーに関する考慮事項

サービス・ソリューションとしてのソフトウェアも含めた IBM ソフトウェア製品 (「ソフトウェア・オファリング」) では、製品の使用に関する情報の収集、エンド・ユーザーの使用感の向上、エンド・ユーザーとの対話またはその他の目的のために、Cookie はじめさまざまなテクノロジーを使用することがあります。多くの場合、ソフトウェア・オファリングにより個人情報が収集されることはありません。 IBM の「ソフトウェア・オファリング」の一部には、個人情報を収集できる機能を持つものがあります。ご使用の「ソフトウェア・オファリング」が、これらの Cookie およびそれに類するテクノロジーを通じてお客様による個人情報の収集を可能にする場合、以下の具体的事項を確認ください。

この「ソフトウェア・オファリング」は、Cookie もしくはその他のテクノロジーを使用して個人情報を収集することはありません。

この「ソフトウェア・オファリング」が Cookie およびさまざまなテクノロジーを使用してエンド・ユーザーから個人を特定できる情報を収集する機能を提供する場合、お客様は、このような情報を収集するにあたって適用される法律、ガイドライン等を遵守する必要があります。これには、エンドユーザーへの通知や同意の要求も含まれますがそれらには限られません。

このような目的での Cookie などの各種テクノロジーの使用について詳しくは、『IBM オンラインでのプライバシー・ステートメントのハイライト』(http://www.ibm.com/privacy/jp/ja/)、『IBM オンラインでのプライバシー・ステートメント』(http://www.ibm.com/privacy/details/jp/ja/) の『クッキー、ウェブ・ビーコン、その他のテクノロジー』というタイトルのセクション、および『IBM Software Products and Software-as-a-Service Privacy Statement』(http://www.ibm.com/software/info/product-privacy) を参照してください。

商標

IBM、IBM ロゴおよび ibm.com は、世界の多くの国で登録された International Business Machines Corp. の商標です。他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。現時点での IBM の商標リストについては、http://www.ibm.com/legal/copytrade.shtml をご覧ください。

INFINIBAND、InfiniBand Trade Association、および INFINIBAND デザイン・マークは、INFINIBAND Trade Association の商標またはサービス・マークです。

Linux は、Linus Torvalds の米国およびその他の国における商標です。

Microsoft、Windows、Windows NT、および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

Java およびすべての Java 関連の商標およびロゴは、Oracle やその関連会社の米国およびその他の国における商標または登録商標です。

UNIX は、The Open Group の米国およびその他の国における登録商標です。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

［サ行］

[タ行]

[ナ行]

[ハ行]

［マ行］

[ヤ行]

[ラ行]

[ワ行]

[数字]

A

B

C

D

E

F

G

I

J

L

M

N

P

R

S

T

U

V

X

Printed in Japan

日本アイ・ビー・エム株式会社
〒103-8510 東京都中央区日本橋箱崎町19-21