Linux on POWER Application Performance Optimization

Matthew Davis

Chakarat Skawratananond

Ramesh Chitor

Nikolay Yevik

IBM (e)Server Enablement

Table of Contents

Introduction

Linux on POWER provides a high performance platform for execution of Linux applications. 64 bit, large memory addresses, world-class vertical scaling and i/o performance are among the features of the POWER architecture, but developers must know how to optimized their applications both in compile time and runtime contexts to take full advantage of the opportunity offered. This document addresses a review of application performance optimization tips, specifically addressing C/C++ compilers, Java, and database performance.

GNU Compiler Collection

Currently, the GNU Compiler Collection is undergoing a massive renovation regarding performance optimization for the POWER architecture. In the upcoming release, GCC 3.4, improved scheduling, signal handling, and architecture specific optimizations (e.g. VMX/Altivec support on PPC970 chips featured in the IBM JS20 BladeCenter) will deliver a dramatic performance advantage to compiled code. However, both SLES8 and RHEL3 currently employ GCC 3.2 versions, and a review of performance optimization for these versions of GCC is provided here.

GCC 3.2 lacks architecture specific optimizations for POWER processors. Therefore, it is not recommended to compile with architecture specific flags, e.g. -mpower. These options are available, but often offer no performance advantage. In contrast to GCC implementations on x86, the -fPIC flag is not implied. In order to generate dynamically linked binaries, -fPIC should be included in all makefiles. This is especially of note to developers porting a codebase from Linux on x86, since this flag may not be explicitly used in existing makefiles.

In addition to awareness of flags not utilized by current releases of GCC for POWER architecture, developers should be aware of common flags available to both x86 and POWER architecture. These include compile flags dealing with relocation, table of contents sizes, floating point options, bit alignment, etc. For a review of these, see standard GCC documentation packaged with source.

The same guildlines for optimization apply to POWER development as elsewhere. When beginning a project or porting effort, use no optimization. Then step to -O2 optimization to take advantage of common optimization routines guaranteed not to vary across architecture. Then step to -O3, using specific flags to accommodate your specific code base.

For further reference, review these documents on GCC development with RHEL and Apple G5 workstations (PPC970 processor), respectively.

http://www.redhat.com/docs/manuals/enterprise/RHEL-3-Manual/gcc/optimize-options.html

http://developer.apple.com/performance/g5optimization.html

IBM VisualAge Compiler Set

VisualAge provides a portfolio of the optimization options tailored to the IBM hardware. For Linux on POWER, applications compiled with VisualAge in many cases have shown significant performance improvements over those compiled with GNU GCC. It should be noted that not all optimizations are beneficial for all applications. A trade-off usually has to be made between the degree of optimization done by the compiler and an increase in compile time accompanied by reduced debugging capability.

Optimization levels

Optimization levels are specified by compiler options. The following table summarizes the compiler behavior at each optimization level.

Option	Behavior
`-qnoopt`	Fast compilation, full debugging support.
`-O2` (same as `-O`)	Performs comprehensive low-level optimization. Only partial debugging is supported.
`-O3`	Performs more extensive optimization than at –O2 and requires large amount of compile time or space. Some precision trade-offs are made.
`-O4`	In addition to –O3, performs Interprocedural analysis, High-order transformations, and Hardware-specific optimization (-qarch=auto, -qtune-=auto, -qcache=auto).
`-O5`	In addition to –O4, performs more detailed Interprocedural analysis.

Recommended approach to using –O2 and –O3:

1. Test and Debug your code without optimization before using –O2.
2. Ensure that your code complies with its language standard. In C code, generic pointers should be char* or void*. All shared variables and pointers to shared variables are marked volatile.
3. Compiles with –O2 as much as possible.
4. Consider using –qalias=noansi rather than turning off optimization if you encounter problems with –O2.
5. Next, use –O3 on as much code as possible.
6. If you encounter problems or performance degradations, consider using –qstrict or –qcompact along with –O3 where necessary.
7. If you still have problems with –O3, switch to –O2 for a subset of files, but consider using –qmaxmem=-1 or –qnostrict, or both.

Optimizing for a particular processor architecture: target machine options

Target machine options are options that instruct the compiler to generate code for optimal execution on a given microprocessor or architecture family. By selecting appropriate target machine options, you can optimize to suit the broadest possible selection of target processors, a range of processors within a given family of processor architectures, or a specific processor. The following options control optimizations affecting individual aspects of the target machine.

Options	Behavior
`-qarch`	Selects a family of processor **architectures** for which instruction code should be generated. The default is `-qarch=ppc`. The following suboptions also available: `auto, pwr3,pwr4, ppc64, ppcgr, rs64b, rs64c`.
`-qtune`	Biases optimization toward execution on a given **microprocessor**, without implying anything about the instruction set architecture to use as a target. The default on Linux is `-qtune=pwr3`. Available suboptions include `auto, pwr3, pwr4, rs64b, rs64c.`
`-qcache`	Defines a specific **cache or memory geometry**. If `-qcache` is used, use `-qhot` or `-qsmp` along with it.

Recommended approach to using target machine options:

To get the most out of target machine options, you should try to specify with `-qarch` the smallest family of machines possible that will be expected to run your code well. Try to specify with `-qtune` the machine where performance should be best. For example, if your application will only be supported on POWER4 systems, use `-O3 -qarch=pwr4 -qtune=pwr4.` Setting –qarch=auto will generate code that may take advantage of instructions available only on the compiling machine. Modification of cache geometry (-qcache) may be useful in cases where the systems have configurable L2 or L3 cache options or where the execution mode reduces the effective size of a shared level of cache (for example, two-core-per-chip SMP execution on POWER4).

POWER platforms support machine instructions not available on other platforms. VisualAge provides a set of built-in functions that directly map to certain POWER instructions. By using these functions, function call return costs, parameter passing, stack adjustment and all the additional costs related with function invocations are eliminated. For the complete list of the supported built-in functions, please see *VisualAge C++ for Linux on pSeries Compiler Reference.*

High-Order transformations (-qhot)

High-Order transformations are optimizations designed to improve the performance of loops through techniques such as interchange, fusion, and unrolling. The option `-qhot=vector` is the default when `-qhot` is specified. When the code is compiled with `-qhot=vector`, some loops are transformed to exploit optimized versions of functions rather than the standard versions.

Recommended approach to using –qhot

Try using -qhot along with -O2 and -O3. It is designed to have a neutral effect when no opportunities for transformation exist. If you experience significantly long compile times or performance degradations with the use of –qhot, try using –qhot=novector, or –qstrict or –qcompact along with –qhot. If necessary, use –qhot selectively, allowing it to improve some of your code.

Interprocedural analysis (-qipa)

With Interprocedural analysis, the compiler performs optimization across different files. It can be specified on the compile step only, or on both compile and link steps ("entire program" mode). Consult the manual for the detailed information of –qipa.

Recommended approach to using –qipa

It is not necessary to compile everything with –qipa.

Always compile main and exported functions with –qipa.

When compiling and linking separately, use –qipa=noobject on the compile step for faster compilation.

Ensure that there is enough space in /tmp (at least 200MB), or use the TMPDIR variable to specify a different directory with sufficient free space.

Try varying level suboption if link time is too long.

To determine if too few or too many functions are inlined, look at the generated code after compiling with –qlist or –qipa=list.

Share-memory parallelism (-qsmp)

Compiling with –qsmp generates the threaded code needed to exploit capability of shared-memory parallel processing supported by some IBM pSeries systems. The default setting of –qsmp is –qsmp=auto:noomp:opt

Recommended approach to using –qsmp

Use –qsmp=omp:noauto if you are using OpenMP, and do not want automatic parallelization.

Before using –qsmp with automatic parallelization, test your programs using optimization and –qhot in a single-threaded manner.

Always use the reentrant compiler invocation (the "_r" invocations) when using –qsmp.

By default, the runtime uses all available processors.

If you are using a dedicated machine or node, consider setting the SPINS and YIELDS variables (suboptions of XLSMPOPTS) to 0. Doing so prevents the operating system from intervening in the scheduling of threads across synchronization boundaries such as barriers.

When debugging an OpenMP program, try using –qsmp=noopt (without –O) to make the debugging information produced by the compiler more precise.

Profile-directed feedback (PDF)

PDF consists of two stages: Stage 1 is a regular compilation using an arbitrary set of optimization options and –qpdf1, that produces an executable or shared object that can be run in a number of different scenarios for an arbitrary amount of time. Stage2 is a recompilation using the same option, except –qpdf2 is used instead of –qpdf1, during which the compiler consumes previously collected data for the purpose of path-biased optimization.

Consult the VisualAge manual for more information.

IBM Java Virtual Machine

At the time of this writing IBM provides JDK 1.3.1 32-bit and JDK 1.4.1, in both 32-bit and 64-bit flavors, for Linux on IBM iSeries and pSeries.

The following discussion applies to IBM JDK 1.3.1 and JDK 1.4.1 for Linux on IBM iSeries and pSeries but specifically targets JDK 1.4.1 SR1 as the latest IBM JDK release at the time of this writing. The IBM JVM Diagnostics Guides for JDK 1.3.1 and JDK 1.4.1 (hereafter JVM Diagnostics Guide) is heavily referenced in this section. It is available in its whole at http://www-106.ibm.com/developerworks/java/jdk/diagnosis/

General guidelines on writing performance-efficient Java code

Avoiding object creation and garbage collection

Whenever possible, creating objects should be avoided to prevent associated performance costs of calling the constructor and subsequent cost of the garbage collecting when an object reaches the end of its lifecycle. Consider these guidelines:

Use the primitive variable types instead of the object types whenever possible. For example, use *int* instead of *Integer*.

Cache frequently used short-lived objects to avoid the need to repeatedly recreate the same objects over and over again and therefore invoke the garbage collector.

When manipulating strings use *StringBuffer* instead of string concatenation due to immutable nature of string objects and therefore the need to create an extra String object that eventually must undergo garbage collection.

Avoid excessive writing to the Java console to reduce the cost of string manipulations, text formatting, and output.

Implement connection pools to the database and reuse connection objects rather than repeatedly open and close connections.

Use thread pooling, that is, avoid incessant creation and discarding of *Thread* objects especially if using threads in abundance.

Avoid calling garbage collector from within your code though *System.gc()* call. Garbage collection is a "Stop the World" event; meaning that all threads of execution will be suspended except for the Garbage Collector threads themselves. If you must call GC do it during non-critical or idle phase.

Avoid allocating objects within loops which keeps the object alive on the Java Heap longer than necessary.

Java Native Interface (JNI)

Writing portions of the application, especially heavily used portions, in native code and linking it with Java is usually intended to improve performance. However, communication between JVM and native code is generally slow, thus too many JNI calls can degrade performance. Therefore, native operations should be grouped together whenever possible to reduce the number of JNI calls.

Handling exceptions natively in the JNI code itself, though unavoidable sometimes, leads to performance degradation. In such cases *ExceptionCheck()* function should be used as it is less computationally expensive than *ExceptionOccurred()*. The latter has to create an object to be referred to as well as a local reference.

Synchronization

To reduce contention in the JVM and operating system use synchronized methods only when feasible. Do not include synchronized methods into a loop structure.

Data structures

As a general rule, avoid using a more complex data structure where simpler one will suffice. For example, instead of vectors use arrays. Use the most efficient way to search and insert elements into a data structure. For example, add and delete from the end of a vector for better performance.

Compilation options to increase performance

Java code compilation

Compile your Java code with *'-O'* optimization flag. Code optimization provides several benefits:

1. obfuscates the code and makes it harder to reverse-engineer
2. significantly enhances source code security;
3. significantly decreases the size of your Java program;
4. improves run-time performance.

JNI code compilation

Compile your JNI C/C++ code using optimization flags intended to increase performance as covered in the section IBM VisualAge Compiler Set above.

Environment settings to increase JVM performance

Spinloop

Currently, adjusting the SPINLOOP variables and the timeslice values sees the biggest performance gains. The IBM_LINUX_SPIINLOOP time is the number of times that a process can spin on a busy lock before blocking. There are 3 SPINLOOP variables available for adjustment (a number from 0 to 100):

IBM_LINUX_SPINLOOP1
IBM_LINUX_SPINLOOP2
IBM_LINUX_SPINLOOP3

The benchmark testing performed on a 16-way LPAR suggests the following settings to be optimal:

IBM_LINUX_SPINLOOP1=96
IBM_LINUX_SPINLOOP2=85
IBM_LINUX_SPINLOOP3=85

As with any other global variable these global variables need to be set in the shell instance where JVM process will run, so that settings can be read by JVM into its global variables table.

Sysctl

As of SLES8, running kernel 2.4.19, there is an option for setting the minimum and maximum for CPU timeslices in the Linux kernel. These are set via the *sysctl* command. It is highly recommended that the *sysctl* value *sched_yield_scale* be set to 1 for Java performance.

Paths

The CLASSPATH variable should have the most often used Java libraries in front of the search path. Same applies to LIBPATH and LD_LIBRARY_PATH variables for most often used JNI shared libraries.

User limits settings

To achieve the best performance it is important that the user that runs JVM process has the user settings appropriately configured. These parameters can be set either temporarily for the duration of login shell session with *ulimit* command or permanently by either adding corresponding *ulimit* statement to one of the files read by login shell (e.g. ~/.profile) or shell specific user resource files; or editing */etc/security/limits.conf*.

Some of the most important settings to be set to *unlimited* as recommended:

Data segment size: *ulimit –d unlimited*
Maximum memory size: *ulimit –m unlimited*
Stack size: *ulimit –s unlimited*
CPU time: *ulimit –t unlimited*
Virtual memory: *ulimit –v unlimited*

For Java applications that do a lot of socket connections and keep them open it is preferable to set the number of file descriptors for a user to a higher than default value by using *ulimit –n* or by setting *nofile* parameter in */etc/security/limits.conf.*

Garbage Collector and Java Heap

Garbage Collector is one of the most important JVM components influencing JVM performance. General IBM JVM discussion, as outlined in IBM JVM Diagnostics Guides for JDK 1.3.1 and JDK 1.4.1, on Garbage Collector and Heap Size tuning applies to IBM JVM on Linux including Linux on POWER with the exception of some IBM JVM on Linux specifics discussed below.

The maximum heap size that is controlled by *–Xmx* can be set to a higher number on 32-bit IBM JVM for Linux than on 32-bit IBM JVM for AIX, due to differences in memory models between the two operating systems. If *–Xmx* option is not specified than the default setting applies - half of the real storage with a minimum of 16 MB and a maximum of 512 MB.

If initial heap size is not specified explicitly with *–Xms* option, it defaults to 4 MB.
For more information on Garbage Collector and Java Heap tuning please see "Debugging Performance Problems: JVM Performance" in the IBM JVM Diagnostics Guides for JDK 1.3.1 and JDK 1.4.1. The chapters "Understanding the Garbage Collector" and "Garbage Collector Diagnostics" in this document may be of additional value.

JIT

JIT is the most important JVM component in terms of performance. For general IBM JVM JIT discussion please refer to the "Understanding the JIT" section of the JVM Diagnostics Guide. For Linux specific details on JIT performance please see "JIT" section of "Linux Problem Determination" chapter and chapter "JIT Diagnostics".

Monitoring JVM

IBM JVM for Linux performance problem determination, JVM monitoring and tools are discussed in detail in the "Linux Problem Determination" chapter of the JVM Diagnostics Guide.

The following chapters may be of additional value:

Tracing Java Applications and the JVM;
Using the JVM monitoring interface (JVMMI);
Using the Reliability, Availability, and Serviceability interface;
Using the JVMPI;
Using third-party tools.

Note on Linux Threading Models and JVM

There are some specifics in threading models implementations that influence JVM performance on different Linux distributions as discussed in the “Linux Problem Determination” chapter of the JVM Diagnostics Guide.

At the time of this writing, IBM JVMs for Linux, including Linux on POWER are not supported on distributions implementing new, enhanced threading library Native POSIX Threads Library for Linux (NPTL).

Another issue to be aware of is thread floating stack limitation on Linux as discussed in “Floating Stacks Limitation” subsection of the JVM Diagnostics Guide.

Note for users of SLES 8 and IBM JDK 1.4.1

Users of SLES 8 Linux distribution should be aware of performance issue with SLES 8 kernel scheduler and JDK 1.4.1 for Linux from IBM due to specifics of SLES 8 scheduler internal implementation. Please read more on the issue in chapter “Linux Problem Determination”, section “Known Limitations on Linux” in the JVM Diagnostics Guide.

IBM DB2

DB2 Utilities

Tuning for IBM DB2 is simplified by the Configuration Advisor wizard, which is run from the Contorl Center. DB2 also provides various utilities, such as RUNSTATS, REORG and REORGCHK, to improve database performance.

Runstats

RUNSTATS is a utility that updates the statistics in the system catalog tables to help with the query optimization process. With these statistics, the information database manager can make decisions that increase the performance of SQL statements. Use RUNSTATS after massive changes to the data and possibly after running REORG.

Reorgchk

REORGCHK examines the data in the system tables and applies formulas to determine whether to reorganize the table and its indexes. REORGCHK can also invoke RUNSTATS before examining the statistics. Run REORGCHK periodically, or when users notice degraded performance.

Reorg

REORG eliminates fragmentation in tables and indexes and may optionally order the rows of a table according to the order of the index. Use the REORG utility after REORGCHK has indicated that REORG is needed. REORG should also be used after performance has suffered following a lengthy succession of data inserts, updates, and deletes, which cause the clustering or space utilization to degrade.

Memory Settings

DB2 agents

Memory allocation for varies for the agent's operating system. Use a minimum of 1 MB for UNIX/Linux and 500 KB for Windows for each DB2 agent. If fenced stored procedures are used, then each user connection has two DB2 agents, in addition to the memory required to run the stored procedure application. The amount of memory required by each agent depends on the nature of the SQL statements performed by the application, such as the number of concurrent cursors opened and the amount of sorting and temp space required. For OLTP applications, there should be less sorting and temp space required and only a handful of concurrent cursors opened at a time.

System Memory

For a 32-bit system, use at least 512 MB of RAM per CPU, up to 4 GB per machine, to support the buffer pools, DB2 agents, and other shared memory objects required for a large number of concurrent users (see Buffer pool size, BUFFPAGE, for more information on buffer pools). More memory may be needed to support applications that run locally or as stored procedures.

For a 64-bit system, the buffer pool can be practically any size. However, for most eCommerce OLTP applications that use a large database, the buffer pool doesn't really need to be more than 8 GB. Bigger is still better, but at some point you'll experience diminishing returns as the buffer pool hit ratio approaches the 98% range. The number of concurrent users (with its impact on the number of agents) determines how much more memory is required.

Oracle 9i Database

Oracle 9i tuning depends heavily on the virtual memory performance of Linux. The virtual memory subsystem is configured using the file */etc/sysctl.conf*. This file can be altered to adjust the virtual memory settings for bdfllush. Below is a recommended setting:

vm.bdflush = 100 1200 128 512 15 5000 500 1884 2

These parameters for bdflush are documented extensively in the Linux kernel documentation at Documentation/sysctl/vm.txt, which is partially reproduced here:

nfract [100]: governs the maximum number of dirty buffers in the buffer cache. Dirty means that the contents of the buffer still have to be written to disk as opposed to a clean buffer, which can just be forgotten about. Setting this to a high value means that Linux can delay disk writes for a long time, but it also means that it will have to do a lot of I/O at once when memory becomes short. A low value will spread out disk I/O more evenly.

ndirty [1200]: gives the maximum number of dirty buffers that bdflush can write to the disk in one time. A high value will mean delayed, bursty I/O, while a small value can lead to memory shortage when bdflush isn't woken up often enough.

nrefill [128]: the number of buffers that bdflush will add to the list of free buffers when refill_freelist() is called. It is necessary to allocate free buffers beforehand, as the buffers often are of a different size than the memory pages, and some bookkeeping needs to be done beforehand. The higher the number, the more memory will be wasted and the less often refill_freelist() will need to run.

refill_freelist [512]: when this comes across more than nref_dirt dirty buffers, it will wake up bdflush.

*age_buffer 50*HZ*, *age_super parameters 5*HZ*: govern the maximum time Linux waits before writing out a dirty buffer to disk. The value is expressed in jiffies (clockticks); the number of jiffies per second is 100. Thus, x*HZ is x seconds. Age_buffer is the maximum age for data blocks, while age_super is for filesystem metadata.

The parameters not discussed explicitly here are not relevant to Oracle performance. For more information on bdflush, see Linux kernel documentation provided in source.

The performance improvements with these bdflush settings were 26% for loads and 7% for TPS.

Network

Network File System

Poor NFS performance is a result of numerous causes. The Optimizing NFS Performance chapter of the Linux NFS How-To at http://nfs.sourceforge.net/nfs-howto/performance.html is considered fundamental reading. Some highlights of this document are outlined here.

RPC Size

Since Linux-2.4.21, RPC size is no longer limited to 8k for NFSv3. The new Limit is 32k. There is some evidence from Linux 2.6 suggesting that larger RPC size increases throughput for sequential reads and writes.

Gigabit Ethernet

If connecting a client and a server through gigabit ethernet, an mtu size of 9000 on both the client and server will increase performance.

TCP vs UDP

TCP performance is better in some cases, UDP is better at others. TCP is recommended for stability; UDP is notoriously unstable for NFS on many platforms.

NFS Threads

8 threads per CPU is the best rule. Testing has shown that a large number of NFSd threads do not significantly affect performance; running 32 treads on a 4-way gives almost the same performance as running 128 treads under normal loads.

NFS increasing

256k per processor is a good number to start with, but experiementation with this number for specific workloads is suggested.

Sync vs Async

Always run sync. NFS async operations are not part of the RFC specifications for NFSv2 or NFSv3.

Miscellaneous Network Tuning

E1000 Adapter

For a detailed documentation see Documentation/networking/e1000.txt in the Linux kernel source.

Interrupt Throttling Rate (ITR)

By default, the driver sets the ITR value dynamically depending on the workload. A completely symmetric Tx/Rx gets a reduced ITR (ITR=2000). A completely asymmetric Tx/Rx runs at ITR=8000, and as the ratio mediates the extremes, the driver adjusts between ITR=2000 and ITR=8000.

During performance analysis work conducted within IBM, setting the ITR rate to 8000 statically improved TCP_RR (response/request) and Full-Duplex performance by around 300%. However, the comes at a trade off of increased CPU utilization.

To adjust the ITR setting on the driver:

ifconfig down all the INTEL gigabit adapters
rmmod e1000
insmod e1000 InterruptThrottlingRate=8000,8000,…

Rx/Tx Descriptors

If the user load is dropping packets on the send or receive side, and memory is not an issue, it is suggested that RxDescriptors and TxDescriptors be set to 1024. This can be checked through "netstat –i" for the Rx fields, but not Tx fields. This is because the e1000 does not account for any Tx errors or drops.

To adjust the Rx/Tx Descriptors:

ifconfig down all the INTEL gigabit adapters
rmmod e1000
insmod e1000 RxDescriptors=1024,1024,… TxDescriptors=1024,1024,…

Kernel Performance Tunning:

txqueuelen

This is the Tx software queue and each Ethernet adapter gets one. The default is 100. This can be checked and configured with *ifconfig*.

If the network performance is suffering as a result of dropped packets, this should be set to 1000 with ifconfig:
ifconfig eth0 txqueuelen=1000

Monitor the txqueuelen for dropped packets using *tc*:

```
tc qdisc add dev etddh0 root pfifo limit 100
tc –s –d qdisc show dev eth0
tc qdisc del dev eth0 root
```

tcp_window_scaling, tcp_timestamps, and tcp_sack

By default, Linux turns these setting on, though they conflict with optimal performance. The MSS (message size) is reported as 1415 on "tcpdump" when it should be 1460. To resolve, it is suggested that the user turn off these three settings:

```
sysctl –w net.ipv4.tcp_window_scaling=0
sysctl –w net.ipv4.tcp_timestamps=0
sysctl –w net.ipv4.tcp_sack=0
```

Note that this is strictly for MTU1500.

Web Applications

For a workload similiar to SPECweb99, the following parameters will improve performance:

ulimit -n 10000 This command sets number of open files; the default is 1024.

Bind only one NIC IRQ per CPU.

Set each NIC TX queue length to 20000 using *ifconfig*; the default is 100.

File systems should be mounted with *noatime* and *nodiratime*. This is for no inode access time updating.

The following kernel arguments augment web application performance:

net.ipv4.nonlocal_bind = 1. This allows processes to bind to non-local IP adresses.
net.ipv4.tcp_timestamps = 0. This turns TCP timestamp support off; the default is on.
net.ipv4.tcp_max_tw_buckets = 2000000. This sets the TCP time-wait buckets pool size; the default is 180000.

net.ipv4.tcp_rmem = 10000000 10000000 10000000. This sets the min/default/max TCP read buffer; the defaults are 4096, 87380, and 174760.
net.ipv4.tcp_wmem = 10000000 10000000 10000000. This sets the min/pressure/max TCP write buffer; the defaults are 4096, 16384, and 131072.
net.ipv4.tcp_mem = 10000000 10000000 10000000. This sets the min/pressure/max TCP buffer space; the defaults are 31744, 32256, and 32768.
net.ipv4.tcp_sack = 0. This turns SACK support off; the default is on.
net.ipv4.tcp_window_scaling = 0. This turns TCP window scaling support off; The default is on.
net.core.hot_list_length = 20000. This is the maximum number of skb-heads to be cached; the default is 128.
net.core.rmem_max = 10000000. This is the maximum receive socket buffer size; the default is 131071.
net.core.wmem_max = 10000000. This is the maximum send socket buffer size; the default is 131071.
net.core.rmem_default = 10000000. This is the default receive socket buffer size; the default is 65535.
net.core.wmem_default = 10000000. This is the default send socket buffer size; the default is 65535.
net.core.optmem_max = 10000000. This is the maximum amount of option memory buffers; the default is 10240.
net.core.netdev_max_backlog = 300000. This is the number of unprocessed input packets before kernel starts dropping them; the default is 300.

Linux Threading Models

The release of the Linux 2.6 kernel in January, 2004 brought with it widespread implementation of the new Native POSIX Threads for Linux (NPTL) threading model. Like it's predecessor, Linux Threads, NPTL is a 1:1 threading model, but due to new efficiency gained in a complete rewrite, NPTL is remarkably faster than the old model. SLES8 Service Pack 3 currently features a 2.4.21 Linux kernel without support for NPTL, though this functionality will arrive in SUSE's soon to be released SLES9 with a 2.6 kernel. In contrast, RHEL3 features a 2.4.21 kernel with NPTL support backported from Linux 2.6. Though this discrepancy is sometimes a thorn with regards to binary compatability between these two Linux distributions, NPTL can provide a windfall of performance. This is especially true with regards to thread intensive Java applications, where testing on NPTL an enabled JVM suggests up to 800% of the expected performance on a Linux Threads system. However, the performance enhancement of NPTL is not limited to Java. Any application that heavily utilizes threading will see a tremendous increase in performance both in terms of startup time and overall performance.

Summary

A platform providing a highly optimized application environment the simultaneous execution of 32 and 64 bit applications across robust vertically and horizontally scalable hardware, Linux on POWER brings the amiable Linux OS to the time tested POWER architecture. Application tuning allows access to the full advantage of this platoform, be it with a choice of compiler sets for C, C++ and Java, tuned enterprise middleware like

IBM DB2, IBM Websphere and Oracle 9i Database Server, traditional high performance UNIX features such as NFS, or cutting edge development from leading Linux distributors.