IBM WebSphere Application Server for z/OS, Version 8.5

Tuning guide

SA32-1077-00

Note

Before using this information, be sure to read the general information under “Notices” on page 75.

Compilation date: June 1, 2012

Contents

How to send your comments

Your feedback is important in helping to provide the most accurate and highest quality information.

- To send comments on articles in the WebSphere Application Server Information Center
  1. Display the article in your Web browser and scroll to the end of the article.
  2. Click on the **Feedback** link at the bottom of the article, and a separate window containing an email form appears.
  3. Fill out the email form as instructed, and submit your feedback.
- To send comments on PDF books, you can email your comments to: **wasdoc@us.ibm.com**.

  Your comment should pertain to specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. Be sure to include the document name and number, the WebSphere Application Server version you are using, and, if applicable, the specific page, table, or figure number on which you are commenting.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer. When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about your comments.

Using this PDF

Links

Because the content within this PDF is designed for an online information center deliverable, you might experience broken links. You can expect the following link behavior within this PDF:

- Links to Web addresses beginning with http:// work.
- Links that refer to specific page numbers within the same PDF book work.
- The remaining links will *not* work. You receive an error message when you click them.

Print sections directly from the information center navigation

PDF books are provided as a convenience format for easy printing, reading, and offline use. The information center is the official delivery format for IBM WebSphere Application Server documentation. If you use the PDF books primarily for convenient printing, it is now easier to print various parts of the information center as needed, quickly and directly from the information center navigation tree.

To print a section of the information center navigation:

1. Hover your cursor over an entry in the information center navigation until the **Open Quick Menu** icon is displayed beside the entry.
2. Right-click the icon to display a menu for printing or searching your selected section of the navigation tree.
3. If you select **Print this topic and subtopics** from the menu, the selected section is launched in a separate browser window as one HTML file. The HTML file includes each of the topics in the section, with a table of contents at the top.
4. Print the HTML file.

For performance reasons, the number of topics you can print at one time is limited. You are notified if your selection contains too many topics. If the current limit is too restrictive, use the feedback link to suggest a preferable limit. The feedback link is available at the end of most information center pages.

Chapter 1. Tuning the Liberty profile

Use this topic to learn about the tunable parameters and attributes of the Liberty profile.

About this task

The Liberty profile support different attributes in the `server.xml` file to influence application performance. You can use these parameters and attributes to achieve better performance.

Procedure

- Tune the JVM.

  Tuning the JVM is a most important tuning step whether you are configuring a development or production environment. When tuning the JVM for the Liberty profile, consider using the `jvm.options` file in the `${server.config.dir}` directory. You can specify each of the JVM arguments that you want to use, one option per line. See Customizing the Liberty profile environment for more information. An example of `jvm.options` file as follows:

  ```
  -Xms50m
  -Xmx256m
  ```

  For a development environment, you might be interested in faster server startup, so consider setting the minimum heap size to a small value, and the maximum heap size to whatever value is needed for your application. For a production environment, setting the minimum heap size and maximum heap size to the same value can provide the best performance by avoiding heap expansion and contraction.

- Tune transport channel services.

  The transport channel services manage client connections, I/O processing for HTTP, thread pools, and connection pools. For applications on the Liberty profile, the following attributes are available for different elements that can be used to improve runtime performance, or scalability, or both. Each of these attributes is also described in Liberty profile: Configuration elements in the server.xml file.

  `maxKeepAliveRequests` of `httpOptions`

  This option specifies the maximum number of persistent requests that are allowed on a single HTTP connection if persistent connections are enabled. A value of *-1* means unlimited. This option can provide benefit to low latency or high throughput applications and SSL connections where building up new a connection can be costly. Here is an example of how you code this option in the `server.xml` file:

  ```
  <httpOptions maxKeepAliveRequests="-1" />
  ```

  `coreThreads` of `executor`

  This option specifies the core number of threads to associate with the executor of the thread pool. The number of threads associated with the executor will quickly grow to this number. If this value is less than `0`, a default value is used. This default value is calculated based on the number of hardware threads on the system.

  Tip: Start your tuning with **`coreThreads`**`="5"` for each hardware thread or logical processor. For example, for a 2-core SMT-4 machine, which represents 8 logical processors, you might use **`coreThreads`**`="40"` as a starting point.

  Here is an example of how you code this option in the `server.xml` file:

  ```
  <executor name="LargeThreadPool" id="default" coreThreads="40" maxThreads="80"
            keepAlive="60s" stealPolicy="STRICT" rejectedWorkPolicy="CALLER_RUNS" />
  ```

  `maxPoolSize` of `connectionManager`

  This option specifies the maximum number of physical connections for the connection pool. The default value is `50`. The optimal setting here depends on the application characteristics. For an application in which every thread obtains a connection to the database, you might start with a 1:1 mapping to the **`coreThreads`** attribute. Here is an example of how you code this option in the `server.xml` file:

```
<connectionManager ... maxPoolSize="40" />
```

purgePolicy of connectionManager

This option specifies which connections to destroy when a stale connection is detected in a pool. The default value is the entire pool. It might be better to purge only the failing connection. Here is an example of how you code this option in the server.xml file:

```
<connectionManager ... purgePolicy="FailingConnectionOnly" />
```

numConnectionsPerThreadLocal of connectionManager

This option specifies the number of database connections to cache for each executor thread. This setting can provide a major improvement on large multicore (8+) machines by reserving the specified number of database connections for each thread.

Using thread local storage for connections can increase performance for applications on multi-threaded systems. When setting **numConnectionsPerThreadLocal** to 1 or more, that number of connections per thread are stored in thread local storage. When using **numConnectionsPerThreadLocal**, two other values need to be considered:

- The number of application threads, and
- The connection pool maximum connections.

For best performance, if you have n applications threads, set maximum pool connections to at least n times the value of **numConnectionsPerThreadLocal** attribute. For example: If you use 20 application threads, then the maximum pool connections should be set to 20 or more; If you set the value of **numConnectionPerThreadLocal** attribute as 2 and there are 20 application threads, then the maximum pool connection must be set to 40 or more. Here is an example of how you code this option in the server.xml file:

```
<connectionManager ... numConnectionsPerThreadLocal="1" />
```

statementCacheSize of dataSource

This option specifies the maximum number of cached prepared statements per connection. This option must be set by reviewing the application code (or an SQL trace gathered from the database or database driver) for all unique prepared statements, and ensuring the cache size is larger than the number of statements. Here is an example of how you code this option in the server.xml file:

```
<dataSource ... statementCacheSize="60" >
```

isolationLevel of dataSource

The datasource isolation level is used to specify the degree of data integrity and concurrency, which in turns controls the level of database locking. Traditionally there are four different options, listed below in order of best performing (least integrity) to worst performing (best integrity).

TRANSACTION_READ_UNCOMMITTED
Dirty reads, non-repeatable reads and phantom reads can occur.

TRANSACTION_READ_COMMITTED
Dirty reads are prevented; non-repeatable reads and phantom reads can occur.

TRANSACTION_REPEATABLE_READ
Dirty reads and non-repeatable reads are prevented; phantom reads can occur.

TRANSACTION_SERIALIZABLE
Dirty reads, non-repeatable reads and phantom reads are prevented.

Here is an example of how you code this option in the server.xml file:

```
<dataSource ... isolationLevel="TRANSACTION_READ_COMMITTED">
```

Chapter 2. Planning for performance

How well a website performs while receiving heavy user traffic is an essential factor in the overall success of an organization. This section provides online resources that you can consult to ensure that your site performs well under pressure.

Procedure

- Consult the following web resources for learning.

  IBM® Patterns for e-Business

  IBM Patterns for e-business is a group of reusable assets that can help speed the process of developing Web-based applications. The patterns leverage the experience of IBM architects to create solutions quickly, whether for a small local business or a large multinational enterprise.

  Planning for availability in the enterprise

  Availability is an achievable service-level characteristic that every enterprise struggles with. The worst case scenario is realized when load is underestimated or bandwidth is overloaded because availability planning was not carefully conducted. Applying the information in this article and the accompanying spreadsheet to your planning exercises can help you avoid such a scenario.

  Hardware configurations for WebSphere® Application Server production environments

  This article describes the most common production hardware configurations, and provides the reasons for choosing each one. It begins with a single machine configuration, and then proceeds with additional configurations that have higher fault tolerance, horizontal scaling, and a separation of web and enterprise bean servers.

- Take advantage of performance functions to improve performance. You can use functions such as balancing workloads with clusters and using the dynamic cache to improve performance.

Application design consideration

This topic describes architectural suggestions for the design and tuning of applications.

The designing applications information contains the architectural suggestions and the implementation of applications. For existing applications, the suggestions might require changing the existing implementations. Tuning the application server and resource parameters can have the greatest effect on performance of the applications that are well designed.

Use designing applications considerations in this topic for tips to ensure your applications are thoughtfully designed and tuned. These considerations include websites and other ideas for finding best practices for designing WebSphere applications, particularly in the realm of WebSphere extensions to the Java Platform, Enterprise Edition (Java EE) specification.

best-practices: Use the following information as an architectural guide when implementing applications:

- Persistence
- Model-view-controller pattern
- Statelessness
- Caching
- Asynchronous considerations
- Third-party libraries

Java EE applications load, store, create, and remove data from relational databases, a process commonly referred to as *persistence*. Most enterprise applications have significant database access. The architecture

and performance of the persistence layer is critical to the performance of an application. Therefore, persistence is a very important area to consider when making architectural choices that require trade-offs related to performance. This guide recommends first focusing on a solution that has clean architecture. The clean architecture considers data consistency, security, maintenance, portability, and the performance of that solution. Although this approach might not yield the absolute peak performance obtainable from manual coding a solution that ignores the mentioned qualities of service, this approach can achieve the appropriate balance of data consistency, maintainability, portability, security, and performance.

Multiple options are available in Java EE for persistence: Session beans using entity beans including container-managed persistence (CMP) or bean-managed persistence (BMP), session beans using Java Database Connectivity (JDBC), and Java beans using JDBC. For the reasons previously mentioned, consider CMP entity persistence because it provides maximum security, maintenance, and portability. CMP is also recommended for good performance. Refer to the Tune the EJB container section of the tuning application servers topic on tuning enterprise beans and more specifically, CMP.

If an application requires using enterprise beans not using EJB entities, the persistence mechanism usually involves the JDBC API. Because JDBC requires manual coding, the Structured Query Language (SQL) that runs against a database instance, it is critical to optimize the SQL statements that are used within the application. Also, configure the database server to support the optimal performance of these SQL statements. Finally, usage of specific JDBC APIs must be considered including prepared statements and batching.

Regardless of which persistence mechanism is considered, use container-managed transactions where the bean delegates management of transactions to the container. For applications that use JDBC, this is easily achieved by using the session façade pattern, which wraps all JDBC functions with a stateless session bean.

Finally, information about tuning the connection over which the EJB entity beans or JDBC communicates can be found in the Tune the data sources section of the tuning application servers topic.

One of the standard Java EE programming architectures is the model-view-controller (MVC) architecture, where a call to a controller servlet might include one or more child JavaServer Pages (JSP) files to construct the view. The MVC pattern is a recommended pattern for application architecture. This pattern requires distinct separation of the view (JSP files or presentation logic), the controller (servlets), and the model (business logic). Using the MVC pattern enables optimization of the performance and scalability of each layer separately.

Implementations that avoid storing the client user state scale and perform the best. Design implementations to avoid storing state. If state storage is needed, ensure that the size of the state data and the time that the state is stored are kept to the smallest possible values. Also, if state storage is needed, consider the possibility of reconstructing the state if a failure occurs, instead of guaranteeing state failover through replication.

Specific tuning of state affects HTTP session state, dynamic caching, and enterprise beans. Refer to the follow tuning guides for tuning the size, replication, and timing of the state storage:
- Session management tuning
- EJB tuning tips
- Tuning dynamic cache with the cache monitor

Most Java EE application workloads have more read operations than write operations. Read operations require passing a request through several topology levels that consist of a front-end web server, the web container of an application server, the EJB container of an application server, and a database. WebSphere Application Server provides the ability to cache results at all levels of the network topology and Java EE programming model that include web services.

Application designers must consider caching when the application architecture is designed because caching integrates at most levels of the programming model. Caching is another reason to enforce the MVC pattern in applications. Combining caching and MVC can provide caching independent of the presentation technology and in cases where there is no presentation to the clients of the application.

Network designers must consider caching when network planning is performed because caching also integrates at most levels of the network topology. For applications that are available on the public Internet, network designers might want to consider Edge Side Include (ESI) caching when WebSphere Application Server caching extends into the public Internet. Network caching services are available in the proxy server for WebSphere Application Server, WebSphere Edge Component Caching Proxy, and the WebSphere plug-in.

Java EE workloads typically consist of two types of operations. You must perform the first type of operation to respond to a system request. You can perform the second type of operation asynchronously after the user request that initiated the operation is fulfilled.

An example of this difference is an application that enables you to submit a purchase order, enables you to continue while the system validates the order, queries remote systems, and in the future informs you of the purchase order status. This example can be implemented synchronously with the client waiting for the response. The synchronous implementation requires application server resources and you wait until the entire operations complete. If the process enables you to continue, while the result is computed asynchronously, the application server can schedule the processing to occur when it is optimal in relation to other requests. The notification to you can be triggered through email or some other interface within the application.

Because the asynchronous approach supports optimal scheduling of workloads and minimal server resource, consider asynchronous architectures. WebSphere Application Server supports asynchronous programming through Java EE Java Message Service (JMS) and message-driven beans (MDB) as well as asynchronous beans that are explained in the Tuning Java Message Service and Tuning MDB topics.

Verify that all the libraries that applications use are also designed for server-side performance. Some libraries are designed to work well within a client application and fail to consider server-side performance concerns, for example, memory utilization, synchronization, and pooling. It is suggested that all libraries that are not developed as part of an application undergo performance testing using the same test methodologies as used for the application.

Additional references:IBM WebSphere Developer Technical Journal: The top 10 Java EE best practicesImprove performance in your XML applications, Part 2

Chapter 3. Taking advantage of performance functions

This topic highlights a few main ways you can improve performance through a combination of product features and application development considerations.

Procedure

- Use one of the following considerations to improve performance.

  Balancing workloads with clusters

  Clusters are sets of servers that are managed together and participate in workload management. The servers that are members of a cluster can be on different host machines, as opposed to the servers that are part of the same node and must be located on the same host machine. A cell can have no clusters, one cluster, or multiple clusters.

  Using the dynamic cache service to improve performance

  The dynamic cache service improves performance by caching the output of servlets, commands, and JavaServer Pages (JSP) files. Dynamic caching features include cache replication among clusters, cache disk offload, Edge-side include caching, and external caching, which is the ability to control caches outside of the application server, such as that of your web server.

- Ensure your applications perform well.

  Take advantage of architectural suggestions and coding best practices to ensure that your applications perform well. See the information about application design considerations and the information on designing applications to learn more about ways you can improve performance of your applications.

Chapter 4. Obtaining advice from the advisors

Advisors provide a variety of recommendations that help improve the performance of your application server.

Before you begin

The advisors provide helpful performance as well as diagnostic advice about the state of the application server.

About this task

Tuning WebSphere Application Server is a critical part of getting the best performance from your website. However, tuning WebSphere Application Server involves analyzing performance data and determining the optimal server configuration. This determination requires considerable knowledge about the various components in the application server and their performance characteristics. The performance advisors encapsulate this knowledge, analyze the performance data, and provide configuration recommendations to improve the application server performance. Therefore, the performance advisors provide a starting point to the application server tuning process and help you without requiring that you become an expert.

The Runtime Performance Advisor is extended to also provide diagnostic advice and is now called the Performance and Diagnostic Advisor. Diagnostic advice provides useful information regarding the state of the application server. Diagnostic advice is especially useful when an application is not functioning as expected, or simply as a means of monitoring the health of application server.

Procedure

- Decide which performance advisor is right for the purpose, Performance and Diagnostic Advisor or Tivoli® Performance Viewer advisor.
- Use the chosen advisor to periodically check for inefficient settings, and to view recommendations.
- Analyze Performance Monitoring Infrastructure data with performance advisors.

Why you want to use the performance advisors

The advisors analyze the Performance Monitoring Infrastructure (PMI) data of WebSphere Application Server using general performance principles, best practices, and WebSphere Application Server-specific rules for tuning. The advisors that are based on this information provide advice on how to set some of your configuration parameters to better tune WebSphere Application Server.

The advisors provide a variety of advice on the following application server resources:

- Object Request Broker service thread pools
- Web container thread pools
- Connection pool size
- Persisted session size and time
- Data source statement cache size
- Session cache size
- Dynamic cache size
- Java virtual machine heap size
- DB2® Performance Configuration wizard
- Connection use violations

For example, consider the data source statement cache. It optimizes the processing of *prepared statements* and *callable statements* by caching those statements that are not used in an active connection. (Both statements are SQL statements that essentially run repeatable tasks without the costs of repeated compilation.) If the cache is full, an old entry in the cache is discarded to make room for the new one. The best performance is generally obtained when the cache is large enough to hold all of the statements that are used in the application. The PMI counter, prepared statement cache discards, indicates the number of statements that are discarded from the cache. The performance advisors check this counter and provide recommendations to minimize the cache discards.

Another example is thread or connection pooling. The idea behind pooling is to use an existing thread or connection from the pool instead of creating a new instance for each request. Because each thread or connection in the pool consumes memory and increases the context-switching cost, the pool size is an important configuration parameter. A pool that is too large can hurt performance as much as a pool that is too small. The performance advisors use PMI information about current pool usage, minimum or maximum pool size, and the application server CPU utilization to recommend efficient values for the pool sizes.

The advisors can also issue diagnostic advice to help in problem determination and health monitoring. For example, if your application requires more memory than is available, the diagnostic adviser tells you to increase the size or the heap for application server.

Performance advisor types and purposes

Two performance advisors are available: the Performance and Diagnostic Advisor and the performance advisor in Tivoli Performance Viewer.

The Performance and Diagnostic Advisor runs in the Java virtual machine (JVM) process of application server; therefore, it does not provide expensive advice. In a stand-alone application server environment, the performance advisor in Tivoli Performance Viewer runs within the application server JVM.

The performance advisor in Tivoli Performance Viewer provides advice to help tune systems for optimal performance and provide recommendations on inefficient settings by using collected Performance Monitoring Infrastructure (PMI) data. Obtain the advice by selecting the performance advisor in Tivoli Performance Viewer.

In a WebSphere Application Server, Network Deployment environment, the performance advisor in Tivoli Performance Viewer runs within the JVM of the node agent and can provide advice on resources that are more expensive to monitor and analyze. The Tivoli Performance Viewer advisor requires that you enable performance modules, counters, or both.

Table 1. Performance and Diagnostic Advisor and Tivoli Performance Viewer advisor. The following chart shows the differences between the Performance and Diagnostic Advisor and the Tivoli Performance Viewer advisor:

	Performance and Diagnostic Advisor	**Tivoli Performance Viewer advisor**
Start location	Application server	Tivoli Performance Viewer client
Invocation of tool	Administrative console	Tivoli Performance Viewer
Output	• The `SystemOut.log` file • The administrative console • JMX notifications	Tivoli Performance Viewer in the administrative console
Frequency of operation	Configurable	When you select refresh in the Tivoli Performance Viewer administrative console

Table 1. Performance and Diagnostic Advisor and Tivoli Performance Viewer advisor (continued). The following chart shows the differences between the Performance and Diagnostic Advisor and the Tivoli Performance Viewer advisor:

	Performance and Diagnostic Advisor	**Tivoli Performance Viewer advisor**
Types of advice	Performance advice: • Object Request Broker (ORB) service thread pools • Web container thread pools • Connection pool size • Persisted session size and time • Prepared statement cache size • Session cache size • Memory leak detection Diagnostic advice: • Connection factory diagnostics • Data source diagnostics Connection usage diagnostics • Detection of connection use by multiple threads • Detection of connection use across components	Performance advice: • ORB service thread pools • Web container thread pools • Connection pool size • Persisted session size and time • Prepared statement cache size • Session cache size • Dynamic cache size • Java virtual machine (JVM) heap size • DB2 Performance Configuration wizard

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Performance and Diagnostic Advisor

Use this topic to understand the functions of the Performance and Diagnostic Advisor.

The Performance and Diagnostic Advisor provides advice to help tune systems for optimal performance and is configured using the WebSphere Application Server administrative console or the wsadmin tool. Running in the Java virtual machine (JVM) of the application server, the Performance and Diagnostic Advisor periodically checks for inefficient settings and issues recommendations as standard product warning messages. These recommendations are displayed both as warnings in the administrative console under Runtime Messages in the WebSphere Application Server Status panel and as text in the application server `SystemOut.log` file. Enabling the Performance and Diagnostic Advisor has minimal system performance impact.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

The Performance and Diagnostic Advisor provides performance advice and diagnostic advice to help tune systems for optimal performance, and also to help understand the health of the system. It is configured

using the WebSphere Application Server administrative console or the wsadmin tool. Running in the Java virtual machine (JVM) of the application server, the Performance and Diagnostic Advisor periodically checks for inefficient settings and issues recommendations as standard product warning messages. These recommendations are displayed as warnings in the administrative console under Runtime Messages in the WebSphere Application Server Status panel, as text in the application server `SystemOut.log` file, and as Java Management Extensions (JMX) notifications. Enabling the Performance and Diagnostic Advisor has minimal system performance impact.

From WebSphere Application Server, Version 6.0.2, you can use the Performance and Diagnostic Advisor to enable the lightweight memory leak detection, which is designed to provide early detection of memory problems in test and production environments.

The advice that the Performance and Diagnostic Advisor gives is all on the server level. The only difference when running in a WebSphere Application Server, Network Deployment environment is that you might receive contradictory advice on resources that are declared at the node or cell level and used at the server level.

For example, two sets of advice are given if a data source is declared at the node level to have a connection pool size of {10,50} and is used by two servers (server1 and server2). If server1 uses only two connections and server2 uses all fifty connections during peak load, the optimal connection pool size is different for the two servers. Therefore, the Performance and Diagnostic Advisor gives two sets of advice (one for server1 and another for server2). The data source is declared at the node level and you must make your decisions appropriately by setting one size that works for both, or by declaring two different data sources for each server with the appropriate level.

Read the using the performance and diagnostic advisor information for startup and configuration steps.

Diagnostic alerts:

In WebSphere Application Server Version 8.5 the Performance and Diagnostic Advisors are extended to provide more diagnostic alerts to help common troubleshoot problems.

Several alerts are made available to monitor connection factory and data sources behavior. Some of these alerts are straightforward and easy to comprehend. Others are much more involved and are intended for use by IBM support only.

ConnectionErrorOccured diagnostic alert

When a resource adapter or data source encounters a problem with connections such that the connection might no longer be usable, it informs the connection manager that a connection error occurred. This causes the destruction of the individual connection or a pool purge, which is the destruction of all connections in the pool, depending on the pool purge policy configuration setting. An alert is sent, indicating a potential problem with the back-end if an abnormally high number of unusable connections are detected.

Connection low-percent efficiency diagnostic alert

If the percentage of time that a connection is used versus held for any individual connections drops beneath a threshold, an alert is sent with a call stack.

Cross-Component Use JCA Programming Model Violation Diagnostic Alert

When you enable cross-component use detection, the application server raises an alert when a connection handle is used by a Java EE application component that is different from the component that originally acquired the handle through a connection factory. This condition might inadvertently occur if an application passes a connection handle in a parameter or an application obtains a handle from a cache that is shared

by multiple application components. If components use a connection handle in this manner, this might result in problems with application or data integrity. Enable the alert to detect the cross-component connection use during development to identify and avoid potential application problems.

Local transaction containment (LTC) nesting threshold exceeded diagnostic alert

For LTC definition, see the Local transaction containment (LTC) and Transaction type and connection behavior information, and Default behavior of managed connections in WebSphere Application Server topic.

If a high number of LTCs are started on a thread before completing, an alert is raised. This alert is useful in debugging some situations where the connection pool is unexpectedly running out of connections due to multiple nested LTCs holding onto multiple shareable connections.

Multi-Thread Use JCA Programming Model Violation Diagnostic Alert

Multi-thread use detection raises an alert when an application component acquires a connection handle using a connection factory, and then the component uses the handle on a different thread from which the handle was acquired. If you use a connection in this manner, this behavior might cause data integrity problems, especially if an application uses a connection handle on a thread that is not managed. Enable the alert to detect multi-thread connection usage during application development.

Pool low-percent efficiency diagnostic alert

If the average time that a connection is held versus used for the all connections in the pool drops beneath a threshold, an alert is sent.

Serial reuse violation diagnostic alert

For information on what serial reuse is, see the transaction type and connection behavior information. Some legitimate scenarios exist, where a serial reuse violation is appropriate, but in most cases this violation is not intended and might lead to data integrity problems.

If this alert is enabled, any time a serial reuse violation occurs within an LTC, an alert is sent.

Surge mode entered or exited diagnostic alert

When surge mode is configured, an alert is sent whenever surge mode engages or disengages. See the surge mode documentation for more information.

Stuck connection block mode entered or exited diagnostic alert

When stuck connection detection is configured, an alert is sent whenever stuck connection blocking starts or stops. See the stuck connection information.

Thread maximum connections exceeded diagnostic alert

When one or more LTCs on a thread ties too many managed connections, or poolable connections for data sources an alert is issued.

Using the Performance and Diagnostic Advisor

The advisors analyze the Performance Monitoring Infrastructure (PMI) data of WebSphere Application Server using general performance principles, best practices, and WebSphere Application Server-specific rules for tuning.

Procedure

1. Ensure that PMI is enabled, which is default. If PMI is disabled, see the enabling PMI using the administrative console information. To obtain advice, you must first enable PMI through the administrative console and restart the server. The Performance and Diagnostic Advisor enables the appropriate monitoring counter levels for all enabled advice when PMI is enabled. If specific counters exist that are not wanted, or when disabling the Performance and Diagnostic Advisor, you might want to disable PMI or the counters that the Performance and Diagnostic Advisor enabled.
2. If running WebSphere Application Server, Network Deployment, you must enable PMI on both the server and the administrative agent, and restart the server and the administrative agent.
3. Click **Servers** > **Application servers** in the administrative console navigation tree.
4. Click *server_name* > **Performance and Diagnostic Advisor Configuration**.
5. Under the **Configuration** tab, specify the number of processors on the server. This setting is critical to ensure accurate advice for the specific configuration of the system.
6. Select the **Calculation Interval**. PMI data is taken over time and averaged to provide advice. The calculation interval specifies the length of time over which data is taken for this advice. Therefore, details within the advice messages display as averages over this interval.
7. Select the **Maximum Warning Sequence**. The maximum warning sequence refers to the number of consecutive warnings that are issued before the threshold is updated. For example, if the maximum warning sequence is set to `3`, then the advisor sends only three warnings, to indicate that the prepared statement cache is overflowing. After three warnings, a new alert is issued only if the rate of discards exceeds the new threshold setting.
8. Specify **Minimum CPU for Working System**. The minimum central processing unit (CPU) for a working system refers to the CPU level that indicates a application server is under production load. Or, if you want to tune your application server for peak production loads that range from 50-90% CPU utilization, set this value to `50`. If the CPU is below this value, some diagnostic and performance advice are still issued. For example, regardless of the CPU level if you are discarding prepared statements at a high rate, you are notified.
9. Specify **CPU Saturated**. The CPU saturated level indicates at what level the CPU is considered fully utilized. The level determines when concurrency rules no longer increase thread pools or other resources, even if they are fully utilized.
10. Click **Apply**.
11. Click **Save**.
12. Click *server_name* > **Performance and Diagnostic Advisor Configuration**.
13. Click the **Runtime** tab.
14. Click **Restart**. Select **Restart** on the Runtime tab to reinitialize the Performance and Diagnostic Advisor using the last configuration information that is saved to disk.

    This action also resets the state of the Performance and Diagnostic Advisor. For example, the current warning count is reset to zero (0) for each message.
15. Simulate a production level load. If you use the Performance and Diagnostic Advisor in a test environment, do any other tuning for performance, or simulate a realistic production load for your application. The application must run this load without errors. This simulation includes numbers of concurrent users typical of peak periods, and drives system resources, for example, CPU and memory, to the levels that are expected in production. The Performance and Diagnostic Advisor provides advice when CPU utilization exceeds a sufficiently high level only. For a list of IBM business partners that provide tools to drive this type of load, see the performance: resource for learning information.
16. Select the check box to enable the Performance and Diagnostic Advisor.

    Tip: To achieve the best results for performance tuning, enable the Performance and Diagnostic Advisor when a stable production-level load is applied.
17. Click **OK**.

18. Select **Runtime Warnings** in the administrative console under the Runtime Messages in the Status panel or look in the SystemOut.log file, which is located in the following directory:

    profile_root/logs/*server_name*

    Some messages are not issued immediately.
19. Update the product configuration for improved performance, based on advice. Although the performance advisors attempt to distinguish between loaded and idle conditions, misleading advice might be issued if the advisor is enabled while the system is ramping up or down. This result is especially likely when running short tests. Although the advice helps in most configurations, there might be situations where the advice hinders performance. Because of these conditions, advice is not guaranteed. Therefore, test the environment with the updated configuration to ensure that it functions and performs better than the previous configuration.

    Over time, the advisor might issue differing advice. The differing advice is due to load fluctuations and the runtime state. When differing advice is received, you need to look at all advice and the time period over which it is issued. Advice is taken during the time that most closely represents the peak production load.

    Performance tuning is an iterative process. After applying advice, simulate a production load, update the configuration that is based on the advice, and retest for improved performance. This procedure is continued until optimal performance is achieved.

What to do next

You can enable and disable advice in the Advice Configuration panel. Some advice applies only to certain configurations, and can be enabled only for those configurations. For example, unbounded Object Request Broker (ORB) service thread pool advice is only relevant when the ORB service thread pool is unbounded, and can only be enabled when the ORB thread pool is unbounded. For more information on Advice configuration, see the advice configuration settings information.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using SystemOut.log , SystemErr.log, trace.log, and activity.log files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Performance and Diagnostic Advisor configuration settings

Use this page to specify settings for the Performance and Diagnostic Advisor.

To view this administrative page, click **Servers** > **Server Types** > **WebSphere application servers** > ***server_name*** > **Performance and Diagnostic Advisor Configuration** under the Performance section.

Enable Performance and Diagnostic Advisor Framework

Specifies whether the Performance and Diagnostic Advisor runs on the server startup.

The Performance and Diagnostic Advisor requires that the Performance Monitoring Infrastructure (PMI) be enabled. It does not require that individual counters be enabled. When a counter that is needed by the Performance and Diagnostic Advisor or is not enabled, the Performance and Diagnostic Advisor enables it automatically. When disabling the Performance and Diagnostic Advisor, you might want to disable Performance Monitoring Infrastructure (PMI) or the counters that Performance and Diagnostic Advisor enabled. The following counters might be enabled by the Performance and Diagnostic Advisor:

- ThreadPools (module)
  - Web Container (module)
    - Pool Size
    - Active Threads

  - Object Request Broker (module)
    - Pool Size
    - Active Threads
- JDBC Connection Pools (module)
  - Pool Size
  - Percent used
  - Prepared Statement Discards
- Servlet Session Manager (module)
  - External Read Size
  - External Write Size
  - External Read Time
  - External Write Time
  - No Room For New Session
- System Data (module)
  - CPU Utilization
  - Free Memory

Enable automatic heap dump collection

Specifies whether the Performance and Diagnostic Advisor automatically generates heap dumps for post analysis when suspicious memory activity is detected.

Calculation Interval

Specifies the length of time over which data is taken for this advice.

PMI data is taken over an interval of time and averaged to provide advice. The calculation interval specifies the length of time over which data is taken for this advice. Details within the advice messages display as averages over this interval. The default value is automatically set to four minutes.

Maximum warning sequence

The maximum warning sequence refers to the number of consecutive warnings that are issued before the threshold is relaxed.

For example, if the maximum warning sequence is set to 3, the advisor only sends three warnings to indicate that the prepared statement cache is overflowing. After three warnings, a new alert is only issued if the rate of discards exceeds the new threshold setting. The default value is automatically set to one.

Number of processors

Specifies the number of processors on the server.

This setting is helpful to ensure accurate advice for the specific configuration of the system. Depending your configuration and system, there may be only one processor utilized. The default value is automatically set to two.

Minimum CPU For Working System

The minimum CPU for working system refers to the point at which concurrency rules do not attempt to free resources in thread pools.

There is a set of concurrency alerts to warn you if all threads in a pool are busy. This can affect performance, and it may be necessary for you to increase them. The CPU bounds are a mechanism to help determine when an application server is active and tunable.

The Minimum CPU for working system sets a lower limit as to when you should consider adjusting thread pools. For example, say you set this value to 50%. If the CPU is less than 50%, concurrency rules *do not* try to free up resources by decreasing pools to get rid of unused threads. That is, if the pool size is 50-100 and only 20 threads are consistently used then concurrency rules would like to decrease the minimum pool size to 20.

CPU Saturated
The CPU Saturated setting determines when the CPU is deemed to be saturated.

There is a set of concurrency alerts to warn you if all threads in a pool are busy. This can affect performance, and it may be necessary for you to increase them. The CPU bounds are a mechanism to help determine when an application server is active and tunable.

The CPU saturated setting determines when the CPU has reached its saturation point. For example, if this is set to 95%, when the CPU is greater than 95% the concurrency rules *do not* try to improve things, that is, increase the size of a thread pool.

Advice configuration settings
Use this page to select the advice you wish to enable or disable.

To view this administrative page, click **Servers** > **Server Types** > **WebSphere application servers** > ***server_name***. Under the Performance section, click **Performance and Diagnostic Advisor Configuration** > **Performance and Diagnostic Advice Configuration**.

Advice name
Specifies the advice that you can enable or disable.

Advice applied to component
Specifies the WebSphere Application Server component to which the advice applies.

Advice type
Categorizes the primary indent of a piece of Advice.

Use Advice type for grouping, and then enabling or disabling sets of advice that is based upon your purpose. Advice has the following types:

- Performance: Performance advice provides tuning recommendations, or identifies problems with your configuration from a performance perspective.
- Diagnostic: Diagnostic advice provide automated logic and analysis relating to problem identification and analysis. These types advice are usually issued when unexpected circumstances are encountered by the application server.

Performance impact
Generalizes the performance overhead that an alert might incur.

The performance impact of a particular piece of advice is highly dependant upon the scenario being run and upon the conditions meet. The performance categorization of alerts is based upon worst case scenario measurements. The performance categorizations are:

- **Low:** Advice has minimal performance overhead. Advice might be run in test and production environments. Cumulative performance overhead is within run to run variance when all advice of this type is enabled.
- **Medium:** Advice has measurable but low performance overhead. Advice might be run within test environments, and might be run within production environments if deemed necessary. Cumulative performance overhead is less than 4% when all advice of this type is enabled.
- **High:** Advice impact is high or unknown. Advice might be run during problem determination tests and functional tests. It is not run in production simulation or production environments unless deemed necessary. Cumulative performance overhead might be significant when all advice of this type is enabled.

Advice status
Specifies whether the advice is stopped, started, or unavailable.

The advice status has one of three values: **Started**, **Stopped** or **Unavailable**.

- **Started**: The advice is enabled.
- **Stopped**: The advice is not enabled.
- **Unavailable**: The advice does not apply to the current configuration, for example, persisted session size advice in a configuration without persistent sessions.

Viewing the Performance and Diagnostic Advisor recommendations
Runtime Performance Advisor uses Performance Monitoring Infrastructure (PMI) data to provide recommendations for performance tuning.

About this task

The Performance and Diagnostic Advisor uses Performance Monitoring Infrastructure (PMI) data to provide recommendations for performance tuning. Running in the Java virtual machine (JVM) of the application server, this advisor periodically checks for inefficient settings, and issues recommendations as standard product warning messages.

Procedure

The Performance and Diagnostic Advisor recommendations are displayed in two locations:

1. The WebSphere Application Server `SystemOut.log` log file.
2. The Runtime Messages panel in the administrative console. To view this administrative page, click **Troubleshooting** > **Runtime Messages** > **Runtime Warning**.

Example

The following log file is a sample output of advice on the `SystemOut.log` file:

```
[4/2/04 15:50:26:406 EST] 6a83e321 TraceResponse W CWTUN0202W:
Increasing the web container thread pool Maximum Size to 48
might improve performance.

Additional explanatory data follows.

Average number of threads: 48.

Configured maximum pool size: 2.

This alert has been issued 1 time(s) in a row.
The threshold will be updated to reduce the
overhead of the analysis.
```

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Starting the lightweight memory leak detection
Use this task to start the lightweight memory leak detection using the Performance and Diagnostic Advisor.

Before you begin

If you have a memory leak and want to confirm the leak, or you want to automatically generate heap dumps on Java virtual machines (JVM) in WebSphere Application Server, consider changing your minimum and maximum heap sizes to be equal. This change provides the memory leak detection more time for reliable diagnosis.

About this task

To start the lightweight memory leak detection using the Performance and Diagnostic Advisor, perform the following steps in the administrative console:

Procedure

1. Click **Servers > Application servers** in the administrative console navigation tree.
2. Click *server_name* **> Performance and Diagnostic Advisor Configuration**.
3. Click the **Runtime** tab.
4. Enable the Performance and Diagnostic Advisor Framework.
5. Click **OK**.
6. From the Runtime or Configuration tab of Performance and Diagnostic Advisor Framework, click **Performance and Diagnostic Advice configuration**.
7. Start the memory leak detection advice and stop any other unwanted advice.

Results

The memory leak detection advice is started.

Important: To achieve the best results for performance tuning, start the Performance and Diagnostic Advisor when a stable production level load is running.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

What to do next

You can monitor any notifications of memory leaks by checking the `SystemOut.log` file or Runtime Messages. For more information, see the “Viewing the Performance and Diagnostic Advisor recommendations” on page 18 topic.

Lightweight memory leak detection

This topic describes memory leaks in Java applications and introduces lightweight memory leak detection.

Although a Java application has a built-in garbage collection mechanism, which frees the programmer from any explicit object deallocation responsibilities, memory leaks are still common in Java applications. Memory leaks occur in Java applications when unintentional references are made to unused objects. This occurrence prevents Java garbage collection from freeing memory.

The term *memory leak* is overused; a memory leak refers to a memory misuse or mismanagement. Old unused data structures might have outstanding references but are never garbage collected. A data structure might have unbounded growth or there might not be enough memory that is allocated to efficiently run a set of applications.

Most existing memory leak technologies are based upon the idea that you know that you have a memory leak and want to find it. Because of these analysis requirements, these technologies have significant performance burdens and are not designed for use as a detection mechanism in production. This limitation means that memory leaks are generally not detected until the problem is critical; the application passes all system tests and is put in production, but it crashes and nobody knows why.

WebSphere Application Server has implemented a lightweight memory leak detection mechanism that runs within the WebSphere Performance and Diagnostic Advisor framework. This mechanism is designed to provide early detection of memory problems in test and production environments. This framework is not designed to provide analysis of the source of the problem, but rather to provide notification and help generating the information that is required to use analysis tools. The mechanism only detects memory leaks in the Java heap and does not detect native leaks.

The lightweight memory leak detection in WebSphere Application Server does not require any additional agents. The detection relies on algorithms that are based on information that is available from the Performance Monitoring Infrastructure service and has minimal performance overhead.

Enabling automated heap dump generation

Use this task to enable automated heap dump generation. This function is not supported when using a Sun Java virtual machine (JVM) which includes WebSphere Application Server running on HP-UX and Solaris operating systems. You need to research taking heap dumps on Sun JVMs or call IBM Support.

Before you begin

Although heap dumps are only generated in response to a detected memory leak, you must understand that generating heap dumps can have a severe performance impact on WebSphere Application Server for several minutes.

About this task

To help you analyze memory leak problems when memory leak detection occurs, some automated heap dump generation support is available.

To enable automated heap dump generation support, perform the following steps in the administrative console:

Procedure

1. Click **Servers > Application servers** in the administrative console navigation tree.
2. Click *server_name* >**Performance and Diagnostic Advisor Configuration**.
3. Click the **Runtime** tab.
4. Select the **Enable automatic heap dump collection** check box.
5. Click **OK**.

Results

The automated heap dump generation support is enabled.

Important: To preserve disk space, the Performance and Diagnostic Advisor does not take heap dumps if more than 10 heap dumps already exist in the WebSphere Application Server home directory. Depending

on the size of the heap and the workload on the application server, taking a heap dump might be quite expensive and might temporarily affect system performance.

The automatic heap dump generation process dynamically reacts to various memory conditions and generates dumps only when it is needed. When the heap memory is too low, the heap dumps cannot be taken or the heap dump generation cannot be complete.

What to do next

You can monitor any notifications of memory leaks by checking the `SystemOut.log` file or Runtime Messages. For more information, see the “Viewing the Performance and Diagnostic Advisor recommendations” on page 18 topic. If a memory leak is detected and you want to find the heap dump, refer to the Locating and analyzing heap dumps topic.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Generating heap dumps manually

Use this task to generate heap dumps manually. This function is not supported on when using a Sun Java virtual machine (JVM) which includes WebSphere Application Server running on HP-UX and Solaris operating systems.

Before you begin

Although heap dumps are generated only in response to a detected memory leak, you must understand that generating heap dumps can have a severe performance impact on WebSphere Application Server for several minutes. When generating multiple heap dumps manually for memory leak analysis, make sure that significant objects are leaked in between the two heap dumps. This approach enables problem determination tools to identify the source of the memory leak.

About this task

You might want to manually generate heap dumps for the analysis of memory leaks. You might also want to designate certain times to take heap dumps because of the overhead involved. On JVM in WebSphere Application Server, you can manually produce heap dumps by using the generateHeapDump operation on WebSphere Application Server managed beans (MBeans) that are special Java beans.

Procedure

1. Determine if you want to use wsadmin or the administrative console to generate your heap dump.
2. To use wsadmin to generate your heap dump, complete the following:
   a. Start the wsadmin scripting client. You have several options to run scripting commands, ranging from running them interactively to running them in a profile.
   b. Invoke the generateHeapDump operation on a JVM MBean, for example:
      - Find a JVM objectName:

        ```
        <wsadmin> set objectName [$AdminControl queryNames
        WebSphere:type=JVM,process=<servername>,node=<nodename>,*]
        ```

      - Invoke the generateHeapDump operation on JVM MBean:

        ```
        <wsadmin> $AdminControl invoke $objectName generateHeapDump
        ```

Table 2. Description of variables. The following table explains variables in the command previously mentioned.

Variable	Description
`$`	is a Jacl operator for substituting a variable name with its value
`invoke`	is the command
`generateHeapDump`	is the operation you are invoking
<servername>	is the name of the server on which you want to generate a heap dump
<nodename>	is the node to which *<servername>* belongs

3. To use the administrative console to generate your heap dump, complete the following:
   a. Start the administrative console.
   b. In the navigation pane, click **Troubleshooting** > **Java dumps and cores**.
   c. Select the *server_name* for which you want to generate the heap dump.
   d. Click **Heap dump** to generate the heap dump for your specified server.

What to do next

After running the **wsadmin** command, the file name of the heap dump is returned. For more information on finding heap dumps, refer to the Locating and analyzing heap dumps topic. When you have a couple of heap dumps, use a number of memory leak problem determination tools to analyze your problem. Memory Dump Diagnostic for Java™ is an offline tool for diagnosing root causes behind memory leaks in the Java heap. See the diagnosing out-of-memory errors and Java heap memory leaks information.

Locating and analyzing heap dumps

Use this task to locate and analyze heap dumps.

Before you begin

Do not analyze heap dumps on the WebSphere Application Server machine because analysis is very expensive. For analysis, transfer heap dumps to a dedicated problem determination machine.

About this task

When a memory leak is detected and heap dumps are generated, you must analyze heap dumps on a problem determination machine and not on the application server because the analysis is very central processing unit (CPU) and disk I/O intensive.

Perform the following procedure to locate the heap dump files.

Procedure

1. On the physical application server where a memory leak is detected, go to the WebSphere Application Server home directory. For example, on the Windows operating system, the directory is:
   profile_root\myProfile
2. IBM heap dump files are usually named in the following way:
   heapdump.*<date>*.*.<timestamp><pid>*.phd
3. Gather all the `.phd` files and transfer them to your problem determination machine for analysis.
4. Many tools are available to analyze heap dumps that include Rational® Application Developer 6.0. WebSphere Application Server serviceability released a technology preview called Memory Dump Diagnostic For Java. You can download this preview from the product download website.

What to do next

When you have a couple of heap dumps, use a number of memory leak problem determination tools to analyze your problem.

Using the performance advisor in Tivoli Performance Viewer

The performance advisor in Tivoli Performance Viewer provides advice to help tune systems for optimal performance and provides recommendations on inefficient settings by using the collected Performance Monitoring Infrastructure (PMI) data.

About this task

Obtain advice by clicking **Performance Advisor** in Tivoli Performance Viewer. The performance advisor in Tivoli Performance Viewer provides more extensive advice than the Performance and Diagnostic Advisor. For example, Tivoli Performance Viewer provides advice on setting the dynamic cache size, setting the Java virtual machine (JVM) heap size and using the DB2 Performance Configuration wizard.

Procedure

1. Enable PMI in the application server.

   To monitor performance data through the PMI interfaces, you must first enable PMI through the administrative console before restarting the server.

   If running in a WebSphere Application Server, Network Deployment environment, you must enable PMI on both the server and on the administrative agent before restarting the server and the administrative agent.
2. Enable data collection and set the PMI monitoring level to Extended.

   The monitoring levels that determine which data counters are enabled can be set dynamically, without restarting the server. These monitoring levels and the data selected determine the type of advice you obtain. The performance advisor in Tivoli Performance Viewer uses the extended monitoring level; however, the performance advisor in Tivoli Performance Viewer can use a few of the more expensive counters (to provide additional advice) and provide advice on which counters can be enabled.

   For example, the advice pertaining to session size needs the PMI statistic set to All. Or, you can use the PMI Custom Monitoring Level to enable the Servlet Session Manager SessionObjectSize counter. The monitoring of the SessionSize PMI counter is expensive, and is not in the Extended PMI statistic set. Complete this action in one of the following ways:

   a. PMI settings.

   b. Enabling Performance Monitoring Infrastructure using the wsadmin tool.
3. In the administrative console, click **Monitoring and Tuning** > **Performance Viewer** > **Current Activity**.
4. Simulate a production level load. Simulate a realistic production load for your application, if you use the performance advisor in a test environment, or do any other performance tuning. The application must run this load without errors. This simulation includes numbers of concurrent users typical of peak periods, and drives system resources, for example, CPU and memory to the levels that are expected in production. The performance advisor only provides advice when CPU utilization exceeds a sufficiently high level. For a list of IBM business partners providing tools to drive this type of load, see the performance: resource for learning information.
5. Log performance data with Tivoli Performance Viewer.
6. Clicking **Refresh** on top of the table of advice causes the advisor to recalculate the advice based on the current data in the buffer.
7. Tuning advice displays when the Advisor icon is chosen in the Tivoli Performance Viewer Performance Advisor. Double-click an individual message for details. Because PMI data is taken over an interval of time and averaged to provide advice, details within the advice message display as averages.

Note: If the Refresh Rate is adjusted, the Buffer Size must also be adjusted to enable sufficient data to be collected for performing average calculations. Currently 5 minutes of data is required. Hence, the following guidelines intend to help you use the Tivoli Performance Advisor:

a. You cannot have a Refresh Rate of more than 300 seconds.

b. RefreshRate * BufferSize > 300 seconds. Buffer Size * Refresh Rate is the amount of PMI data available in memory and it must be greater than 300 seconds.

c. For the Tivoli Performance Advisor to work properly with Tivoli Performance Viewer logs, the logs must be at least 300 seconds of duration.

For more information about configuring user and logging settings of Tivoli Performance Viewer, refer to the configuring Tivoli Performance Viewer settings information.

8. Update the product configuration for improved performance, based on advice. Because Tivoli Performance Viewer refreshes advice at a single instant in time, take the advice from the peak load time. Although the performance advisors attempt to distinguish between loaded and idle conditions, misleading advice might be issued if the advisor is enabled while the system is ramping up or down. This result is especially likely when running short tests. Although the advice helps in most configurations, there might be situations where the advice hinders performance. Because of these conditions, advice is not guaranteed. Therefore, test the environment with the updated configuration to ensure it functions and performs well.

   Over a period of time the advisor might issue differing advice. The differing advice is due to load fluctuations and run-time state. When differing advice is received, you need to look at all advice and the time period over which it was issued. You must take advice during the time that most closely represents the peak production load.

   Performance tuning is an iterative process. After applying advice, simulate a production load, update the configuration that is based on the advice, and retest for improved performance. This procedure is continued until optimal performance is achieved.

Chapter 5. Tuning the application serving environment

Use this topic to understand the benefits of tuning for optimal performance. Learn about the tunable parameters of the major WebSphere Application Server components and how these parameters affect performance.

Before you begin

WebSphere Application Server provides tunable settings for its major components so that you can adjust the runtime environment to match the characteristics of your application. Applications can run successfully without changing the default values for these tuning parameters. Other applications might need changes, for example, a larger heap size, to achieve optimal performance.

Performance tuning can yield significant gains in performance even if an application is not optimized for performance. However, correcting shortcomings of an application typically results in higher performance gains than are possible with just altering tuning parameters. Many factors contribute to a high performing application.

About this task

- For your convenience, procedures for tuning parameters in other products, such as DB2, web servers and operating systems are included. Because these products might change, consider these descriptions as suggestions.

  Each WebSphere Application Server process has parameters that influence application performance. You can use the WebSphere Application Server administrative console to configure and tune applications, web containers, Enterprise JavaBeans (EJB) containers, application servers, and nodes in the administrative domain.

Procedure

1. Run the **applyPerfTuningTemplate.py** script as the starting point for improving the performance of a specific application server. This python-based tuning script, along with one of its template files, applies the recommended performance tuning settings for a typical development, production, or environment that is ready for immediate use. The **applyPerfTuningTemplate.py** script, and its associated templates and properties files, are located in the *WAS_HOME*/bin directory.
2. Use the performance advisors, the suggested procedures or parameters in the tuning parameter hot list, and the information on troubleshooting performance problems to optimize your WebSphere Application Server instances to their fullest extent.

   Performance advisors
   The performance advisors use the Performance Monitoring Infrastructure (PMI) data to suggest configuration changes to Object Request Broker (ORB) service thread pools, web container thread pools, connection pool size, persisted session size and time, prepared statement cache size, and session cache size. The Runtime Performance Advisor runs in the application server process, while the other advisor runs in the Tivoli Performance Viewer. For more information, see the documentation about using the Performance and Diagnostic Advisor and use the performance advisor in Tivoli Performance Viewer.

   Tuning parameter hot list
   Review the documentation about the tuning parameter hot list. These parameters have an important impact on performance. Because these parameters are application-dependent, the parameter settings for specific applications and environments can vary.

   Tuning parameter index for z/OS®
   Performance tuning for WebSphere Application Server for z/OS operating systems becomes complex because the nature of the runtime environment involves many different components of the operating system and middleware.

To find information and parameters for tuning the z/OS operating system, subsystems, the WebSphere Application Server for z/OS runtime environment, and some Java 2 Platform, Enterprise Edition (Java EE) application tuning tips, see the documentation about the tuning parameter hot list.

Note: You can read the WebSphere Application Server for z/OS tuning guidelines, which will explain how to tune the middleware. However, it is important that you ensure that your application is optimally designed to improve performance. Often, poorly written or designed application code changes can have a significant effect on overall performance.

If you are a WebSphere Application Server administrator or system programmer on WebSphere Application Server for z/OS, see the documentation about the tuning index for WebSphere Application Server for z/OS. Each parameter description explains the parameter; provides reasons to adjust the parameter; describes how to view or set the parameter; and indicates default and recommended values.

Troubleshooting performance

To save you time detecting problems and help you troubleshoot performance problems, see the documentation about troubleshooting performance.

Tuning parameter hot list

The following hot list contains recommendations that have improved performance or scalability, or both, for many applications.

WebSphere Application Server provides several tunable parameters and options to match the application server environment to the requirements of your application.

- Review the hardware and software requirements

  It is critical for proper functionality and performance to satisfy the minimum hardware and software requirements. Refer to IBM WebSphere Application Server supported hardware, software, and APIs website which details hardware and software requirements.
- Check hardware configuration and settings

  Verify network interconnections and hardware configuration is setup for peak performance.
- Tune the operating systems

  Operating system configuration plays a key role in performance. For example, adjustments such as TCP/IP parameters might be necessary for your application
- Set the minimum and maximum Java virtual machine (JVM) heap sizes

  Many applications need a larger heap size then the default for best performance. It is also advised to select an appropriate GC policy based on the application's characteristics.
- Use a type 4 (or pure Java) JDBC driver

  In general, the type 2 JDBC driver is recommended if the database exists on the same physical machine as the WebSphere instance. However, in the case where the database is in a different tier, the type 4 JDBC driver offers the fastest performance since they are pure Java not requiring native implementation. Use the previous link to view a list of database vendor-specific requirements, which can tell you if a type 4 JDBC driver is supported for your database.

  See the *Administering applications and their environment* PDF for more information.
- Tune WebSphere Application Server JDBC data sources and associated connection pools

  The JDBC data source configuration might have a significant performance impact. For example, the connection pool size and prepared statement cache need to be sized based on the number of concurrent requests being processed and the design of the application.

  See the *Administering applications and their environment* PDF for more information.
- Ensure that the transaction log is assigned to a fast disk

  Some applications generate a high rate of writes to the WebSphere Application Server transaction log. Locating the transaction log on a fast disk or disk array can improve response time

See the *Administering applications and their environment* PDF for more information.

- Tune related components, for example, database

  In many cases, some other component, for example, a database, needs adjustments to achieve higher throughput for your entire configuration.

  For more information, see the *Administering applications and their environment* PDF for more information.
- Disable functions that are not required

  For example, if your application does not use the web services addressing (WS-Addressing) support, disabling this function can improve performance.

  Attention: Use this property with care because applications might require WS-Addressing MAPs to function correctly. Setting this property also disables WebSphere Application Server support for the following specifications, which depend on the WS-Addressing support: Web Services Atomic Transactions, Web Services Business Agreement and Web Services Notification.

  To disable the support for WS-Addressing, refer to Enabling Web Services Addressing support for JAX-RPC applications
- Tuning index

  One of the goals of the product programming model and runtime is to significantly simplify the work required for application developers to write and deploy applications. Sometimes we say that the product relieves the application programmer of many of the plumbing tasks involved in developing applications. For example, application code in the product does not concern itself directly with remote communication--it locates objects which may be local or remote and drives methods. Therefore, you won't see any direct use of socket calls or TCP/IP programming in this application code.

  This separation of what you want to do from where you do it is one aspect of removing the application programmers from plumbing tasks. Other considerations are not having to deal with data calls for some types of beans, potentially user authentication, and threading. There are generally no calls from the application code to touch sockets, RACF® calls, or management of threading. Removing this from the application programmer doesn't mean this work won't get done. Rather, it means that there may be more work for the DBA, the network administrator, the security administrator, and the performance analyst.

  There are four layers of tuning that need to be addressed:
  - “Tuning the z/OS operating system” on page 50
  - Tuning the subsystems
  - Tuning the product runtime
  - Tuning for J2EE applications

  We deal with the first three in separate sections under this article and briefly touch on the fourth. For more information on tuning applications, refer to Using application clients.
- Tuning the subsystems

  Steps involved in tuning the z/OS subsystems to optimize product performance include:
  - DB2 tuning tips for z/OS
  - Security tuning tips
  - “Tuning TCP/IP buffer sizes” on page 30
  - Tuning GRS when using global transactions in z/OS
  - “Tuning the IBM virtual machine for Java” on page 31
  - Tuning for applications that access CICS® in z/OS
- Tuning the product runtime

  Steps involved in tuning the product runtime to optimize performance include reviewing the:
  - Review the product configuration
  - Internal tracing tips for the product
  - Location of executable programs tips for z/OS
  - Security tuning tips
- Review the product configuration

The first thing to do is review the product configuration. One simple way to do this is to look in your application control and server regions in SDSF. When each server starts, the runtime prints out the current configuration data in the job log.

- Internal tracing tips for the product

  Product traces can be extremely helpful in detecting and diagnosing problems. By properly setting trace options, you can capture the information needed to detect problems without significant performance overhead.

  – Verify that you are not collecting more diagnostic data than you need.

    You should check your tracing options to verify that the ras_trace_defaultTracingLevel property is set to `0` or `1`, and that the ras_trace_basic, and ras_trace_detail properties are not set.

    How to view or set: In the administrative console:
    1. Click **Environment** > **WebSphere variables**.
    2. On the Configuration Tab check for any of these properties in the name field and observe their settings in the value field.
    3. To change the setting for one of these properties, click the property name the name field, and then specify the new setting in the value field. You can also describe the setting in the description field on this tab.
    4. To add one of these properties, click **New**, and then specify the property name the name field, and the property setting in the value field.

  – If you use any level of tracing, including ras_trace_defaultTracingLevel=1, verify that the ras_trace_outputLocation property is set to BUFFER.

    When the ras_trace_defaultTracingLevel property is set to `1` exceptions are written to the trace log as well as to the ERROR log.

    - It is best to trace to CTRACE.

      If you are tracing to SYSPRINT with ras_trace_defaultTracingLeve set to `3`, you might experience an almost 100% throughput degradation. If you are tracing to CTRACE, however, you might only experience a 15% degradation in throughput.

  – Set the ras_trace_BufferCount property to `4` and the ras_trace_BufferSize property to `128`.

    This setting reserves 512 KB of storage for the trace buffers, which is the minimum amount of storage that is allowed, and reduces memory requirements.

  – Disable JRAS tracing.

    To disable JRAS tracing, look for the following lines in the trace.dat file that is pointed to by the JVM properties file:

    ```
    com.ibm.ejs.*=all=disable

    com.ibm.ws390.orb=all=disable
    ```

    Verify that both lines are set to `disable`, or delete the two lines.

    Note: If a value is specified for the ras_trace_outputLocation property, you might be tracing and not know it.

- Location of executable programs tips

  The next thing to review in the configuration is where your program code is located. IBM recommends that you install as much of the product code in LPA as is reasonable. This ensures that you have eliminated any unnecessary steplibs which can adversely affect performance. If you must use STEPLIBs, verify that any `STEPLIB DD` in the controller and servant procs do not point to any unnecessary libraries. Refer to “UNIX System Services (USS) tuning tips for z/OS” on page 54 for USS shared file system tuning considerations.

  If you choose to not put most of the runtime in LPA, you might find that your processor storage gets a bigger workout as the load increases. At a minimum, the product starts three address spaces, so that any code that is not shared loads three copies rather than one. As the load increases, many more servants might start and contribute additional load on processor storage.

  Review the `PATH` statement to ensure that only required programs are in the PATH and that the order of the PATH places frequently-referenced programs in the front.

- Tuning for J2EE applications

  Steps involved in tuning the J2EE applications performance include:
  –
  – "Tuning TCP/IP buffer sizes" on page 30
  – Tuning for SOAP
  – "Tuning message-driven bean processing on z/OS by using WebSphere MQ as the messaging provider in ASF mode" on page 60
- Review your application design

  You can track many performance problems back to the application design. Review the design to determine if it causes performance problems.

Directory conventions

References in product information to *app_server_root*, *profile_root*, and other directories imply specific default directory locations. This article describes the conventions in use for WebSphere Application Server.

Default product locations - z/OS

app_server_root
: Refers to the top directory for a WebSphere Application Server node.

  The node may be of any type—application server, deployment manager, or unmanaged for example. Each node has its own *app_server_root*. Corresponding product variables are *was.install.root* and *WAS_HOME*.

  The default varies based on node type. Common defaults are *configuration_root*/AppServer and *configuration_root*/DeploymentManager.

configuration_root
: Refers to the mount point for the configuration file system (formerly, the configuration HFS) in WebSphere Application Server for z/OS.

  The *configuration_root* contains the various *app_server_root* directories and certain symbolic links associated with them. Each different node type under the *configuration_root* requires its own cataloged procedures under z/OS.

  The default is /wasv8config/*cell_name*/*node_name*.

plug-ins_root
: Refers to the installation root directory for Web Server Plug-ins.

profile_root
: Refers to the home directory for a particular instantiated WebSphere Application Server profile.

  Corresponding product variables are *server.root* and *user.install.root.*

  In general, this is the same as *app_server_root*/profiles/*profile_name*. On z/OS, this will always be *app_server_root*/profiles/default because only the profile name "default" is used in WebSphere Application Server for z/OS.

smpe_root
: Refers to the root directory for product code installed with SMP/E or IBM Installation Manager.

  The corresponding product variable is *smpe.install.root*.

  The default is /usr/lpp/zWebSphere/V8R5.

Tuning TCP/IP buffer sizes

WebSphere Application Server uses the TCP/IP sockets communication mechanism extensively. For a TCP/IP socket connection, the send and receive buffer sizes define the receive window. The receive window specifies the amount of data that can be sent and not received before the send is interrupted. If too much data is sent, it overruns the buffer and interrupts the transfer. The mechanism that controls data transfer interruptions is referred to as flow control. If the receive window size for TCP/IP buffers is too small, the receive window buffer is frequently overrun, and the flow control mechanism stops the data transfer until the receive buffer is empty.

About this task

TCP/IP can be the source of some significant remote method delays.

To change the system wide value, perform the following steps:

Procedure

Tune the TCP/IP buffer sizes.

1. First, ensure that you have defined enough sockets to your system and that the default socket timeout of 180 seconds is not too high. To allow enough sockets, update the BPXPRMxx parmlib member:
   a. Set MAXSOCKETS for the AF_INET filesystem high enough.
   b. Set MAXFILEPROC high enough.

   Note:

   You should set MAXSOCKETS and MAXFILEPROC to at least 5000 for low-throughput, 10000 for medium-throughput, and 35000 for high-throughput WebSphere transaction environments. Setting high values for these parameters should not cause excessive use of resources unless the sockets or files are actually allocated.

   Example:

```
/* Open/MVS Parmlib Member                                          */
/* CHANGE HISTORY:                                                  */
/*   01/31/02 AEK Increased MAXSOCKETS on AF_UNIX from 10000 to 50000*/
/*                per request from My Developer                     */
/*   10/02/01 JAB Set up shared HFS                                 */

/* KERNEL RESOURCES              DEFAULT             MIN MAX         */
/* ========================      ================== === =========== */
   .
   .
   MAXFILEPROC(65535)            /* 64               3   65535       */

   .
   .
    NETWORK DOMAINNAME(AF_INET) DOMAINNUMBER(2) MAXSOCKETS(30000)
   .
```

2. Next check the specification of the port in TCPIP profile dataset to ensure that NODELAYACKS is specified as follows:

   ```
   PORT 8082 TCP NODELAYACKS
   ```

   In your runs, changing this could improve throughput by as much as 50% (this is particularly useful when dealing with trivial workloads). This setting is important for good performance when running SSL.
3. You should ensure that your DNS configuration is optimized so that lookups for frequently-used servers and clients are being cached.

Caching is sometimes related to the name server's Time To Live (TTL) value. On the one hand, setting the TTL high will ensure good cache hits. However, setting it high also means that, if the Daemon goes down, it will take a while for everyone in the network to be aware of it.

A good way to verify that your DNS configuration is optimized is to issue the **oping** and **onslookup** USS commands. Make sure they respond in a reasonable amount of time. Often a misconfigured DNS or DNS server name will cause delays of 10 seconds or more.

4. Increase the size of the TCPIP send and receive buffers from the default of 16K to at least 64K. This is the size of the buffers including control information beyond what is present in the data that you are sending in your application. To do this specify the following:

```
TCPCONFIG TCPSENDBFRSIZE 65535
          TCPRCVBUFRSIZE 65535
```

gotcha: It is unreasonable, in some cases, to specify 256 KB buffers.

5. Increase the default listen backlog.

Note:

The default listen backlog is used to buffer spikes in new connections which come with a protocol like HTTP. The default listen backlog is 10 requests. You should use the TCP transport channel listenBacklog custom property to increase this value to something larger. For example:

```
listenBacklog=100
```

6. Reduce the finwait2 time.

In the most demanding benchmarks you may find that even defining 65K sockets and file descriptors does not give you enough 'free' sockets to run 100%. When a socket is closed abnormally (for example, no longer needed) it is not made available immediately. Instead it is placed into a state called finwait2 (this is what shows up in the netstat -s command). It waits there for a period of time before it is made available in the free pool. The default for this is 600 seconds.

gotcha: Unless you have trouble using up sockets, you should leave this set to the default value. If you are using z/OS Version 1.2 or above, you can control the amount of time the socket stays in finwait2 state by specifying the following in the configuration file:

```
FINWAIT2TIME 60
```

Results

Repeat this process until you determine the ideal buffer size.

What to do next

The TCP/IP buffer sizes are changed. Repeat this process until you determine the ideal buffer size.

Tuning the JVM

Tuning the IBM virtual machine for Java

An application server is a Java based server and requires a Java virtual machine (JVM) environment to run and support the enterprise applications that run on it. As part of configuring your application server, you can configure the Java SE Runtime Environment to tune performance and system resource usage. This topic applies to IBM virtual machines for Java.

Before you begin

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

- Determine the type of JVM on which your application server is running.

  Issue the **java –fullversion** command from within your application server *app_server_root*/`java/bin` directory.

  In response to this command, the Java writes information about the JVM, including the JVM provider information, to stderr.
- Verify that the most recent service update is installed on your system. Almost every new service level includes JVM performance improvements.

About this task

Each JVM vendor provides detailed information on performance and tuning for their JVM. Use the information provided in this topic in conjunction with the information that is provided with the JVM that is running on your system.

Both the controller and the servant contain a JVM. The information contained in this topic applies only to the JVM in the servant. Typically, the JVM in the controller does not need to be tuned.

A Java SE Runtime Environment provides the environment for running enterprise applications and application servers. Therefore the Java configuration plays a significant role in determining performance and system resource consumption for an application server and the applications that run on it.

The IBM virtual machine for Java Version 6.0 includes the latest in Java Platform, Enterprise Edition (Java EE) specifications, and provides performance and stability improvements over previous versions of Java.

Even though JVM tuning is dependent on the JVM provider you use, there are some general tuning concepts that apply to all JVMs. These general concepts include:

- Compiler tuning. All JVMs use Just-In-Time (JIT) compilers to compile Java byte codes into native instructions during server runtime.
- Java memory or heap tuning. Tuning the JVM memory management function, or garbage collection, is a good starting point for improving JVM performance.
- Class loading tuning.
- Start up versus runtime performance optimization

The following steps provide specific instructions on how to perform the following types of tuning for each JVM. The steps do not have to be performed in any specific order.

Procedure

1. Limit the number of dumps that are taken in specific situations.

   In certain error conditions, multiple application server threads might fail and the JVM requests a TDUMP for each of those threads. If a significant number of threads fail at the same time, the resulting number of TDUMPs that are taken concurrently might lead to other system problems, such. as a shortage of auxiliary storage. Use the JAVA_DUMP_OPTS environment variable to specify the number of dumps that you want the JVM to produce in certain situations. The value specified for this variable

does not affect the number of TDUMPS that are generated because of com.ibm.jvm.Dump.SystemDump() calls from applications that are running on the application server.

For example, if you want to configure JVM such that it:

- Limits the number of TDUMPs that are taken to one
- Limits the number of JAVADUMPs taken to a maximum of three
- Does not capture any documentation if an INTERRUPT occurs

Then, set the JAVA_DUMP_OPTS variable to the following value:

```
JAVA_DUMP_OPTS=ONANYSIGNAL(JAVADUMP[3],SYSDUMP[1]),ONINTERRUPT(NONE)
```

2. Optimize the startup and runtime performance.

   In some environments, such as a development environment, it is more important to optimize the startup performance of your application server rather than the runtime performance. In other environments, it is more important to optimize the runtime performance. By default, IBM virtual machines for Java are optimized for runtime performance, while HotSpot-based JVMs are optimized for startup performance.

   The Java Just-in-Time (JIT) compiler impacts whether startup or runtime performance is optimized. The initial optimization level that the compiler uses influences the length of time that is required to compile a class method, and the length of time that is required to start the server. For faster startups, reduce the initial optimization level that the compiler uses. However if you reduce the initial optimization level, the runtime performance of your applications might decrease because the class methods are now compiled at a lower optimization level.

   - **-Xquickstart**

     This setting influences how the IBM virtual machine for Java uses a lower optimization level for class method compiles. A lower optimization level provides for faster server startups, but lowers runtime performance. If this parameter is not specified, the IBM virtual machine for Java defaults to starting with a high initial optimization level for compiles, which results in faster runtime performance, but slower server starts.

     You can set this property on the Java virtual machine panel using the administrative console. For details, read the information about Java virtual machine settings.

Information	Value
Default	High initial compiler optimization level
Recommended	High initial compiler optimization level
Usage	Specifying `-Xquickstart` improves server startup time.

   To speed up JVM initialization and improve server startup time, specify the following command-line arguments in the **General JVM arguments** field in the General Properties section of the Configuration Tab.

```
-Xquickstart
-Xverify:none
```

3. Configure the heap size.

   The Java heap parameters influence the behavior of garbage collection. Increasing the heap size supports more object creation. Because a large heap takes longer to fill, the application runs longer before a garbage collection occurs. However, a larger heap also takes longer to compact and causes garbage collection to take longer.

   The JVM uses defined thresholds to manage the storage that it is allocated. When the thresholds are reached, the garbage collector is invoked to free up unused storage. Therefore, garbage collection can cause significant degradation of Java performance. Before changing the initial and maximum heap sizes, you should consider the following information:

   - In the majority of cases you should set the maximum JVM heap size to a value that is higher than the initial JVM heap size. This setting allows for the JVM to operate efficiently during normal, steady state periods within the confines of the initial heap. This setting also allows the JVM to operate

effectively during periods of high transaction volume because the JVM can expand the heap up to the value specified for the maximum JVM heap size. In some rare cases, where absolute optimal performance is required, you might want to specify the same value for both the initial and maximum heap size. This setting eliminates some overhead that occurs when the JVM expands or contracts the size of the JVM heap. Before changing any of the JVM heap sizes, verify that the JVM storage allocation is large enough to accommodate the new heap size.

- Do not make the size of the initial heap so large that while it initially improves performance by delaying garbage collection, when garbage collection does occur, the collection process affects response time because the process has to run longer.

Java heap information is contained in SMF records and can be viewed dynamically using the **DISPLAY,JVMHEAP** console command.

To use the administrative console to configure the heap size:

a. In the administrative console, click **Servers > Server Types > WebSphere application servers >** *server_name*.
b. In the Server Infrastructure section, click **Java and process management > Process definition**.
c. Select either **Control** or **Servant**, and then select **Java virtual machine**.
d. Specify a new value in either the **Initial heap size** or the **Maximum heap size** field.

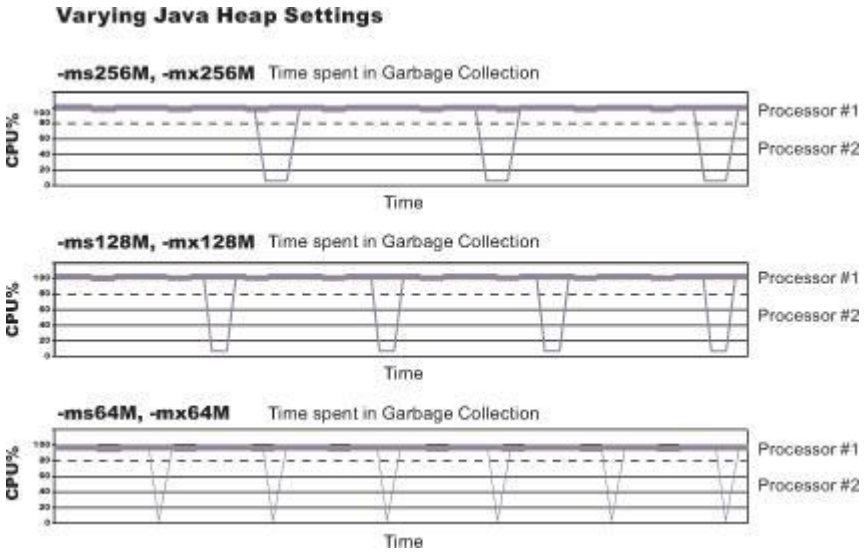
   You can also specify values for both fields if you need to adjust both settings.

   For performance analysis, the initial and maximum heap sizes should be equal.

   The Initial heap size setting specifies, in megabytes, the amount of storage that is allocated for the JVM heap when the JVM starts. The Maximum heap size setting specifies, in megabytes, the maximum amount of storage that can be allocated to the JVM heap. Both of these settings have a significant effect on performance.

   If you are tuning a production system where you do not know the working set size of the enterprise applications that are running on that system, an appropriate starting value for the initial heap size is 25 percent of the maximum heap size. The JVM then tries to adapt the size of the heap to the working set size of the application.

   The following illustration represents three CPU profiles, each running a fixed workload with varying Java heap settings. In the middle profile, the initial and maximum heap sizes are set to 128 MB. Four garbage collections occur. The total time in garbage collection is about 15 percent of the total run. When the heap parameters are doubled to 256 MB, as in the top profile, the length of the work time increases between garbage collections. Only three garbage collections occur, but the length of each garbage collection is also increased. In the third profile, the heap size is reduced to 64 MB and exhibits the opposite effect. With a smaller heap size, both the time between garbage collections and the time for each garbage collection are shorter. For all three configurations, the total time in garbage collection is approximately 15 percent. This example illustrates an important concept about the Java heap and its relationship to object utilization. A cost for garbage collection always exists when running enterprise applications.



Run a series of tests that vary the Java heap settings. For example, run experiments with 128 MB, 192 MB, 256 MB, and 320 MB. During each experiment, monitor the total memory usage. If you expand the heap too aggressively, paging can occur.

If paging occurs, reduce the size of the heap or add more memory to the system.

When all the runs are finished, compare the following statistics:

- Number of garbage collection calls
- Average duration of a single garbage collection call
- Ratio between the length of a single garbage collection call and the average time between calls

If the application is not over utilizing objects and has no memory leaks, the state of steady memory utilization is reached. Garbage collection also occurs less frequently and for short duration.

If the heap free space settles at 85 percent or more, consider decreasing the maximum heap size values because the application server and the application are under-utilizing the memory allocated for heap.

If you have servers configured to run in 64-bit mode, you can specify a JVM maximum heap size for those servers that is significantly larger than the default setting. For example, you can specify an initial maximum heap size of 1844 MB for the controller and the servant if the server is configured to run in 64-bit mode.

e. Click **Apply**.

f. Click **Save** to save your changes to the master configuration.

g. Stop and restart the application server.

You can also use the following command-line parameters to adjust these settings. These parameters apply to all supported JVMs and are used to adjust the minimum and maximum heap size for each application server or application server instance.

- **-Xms**

  This parameter controls the initial size of the Java heap. Tuning this parameter reduces the overhead of garbage collection, which improves server response time and throughput. For some applications, the default setting for this option might be too low, which causes a high number of minor garbage collections.

Information	Value
Default	50 MB
Recommended	Workload specific, but higher than the default.
Usage	Specifying `-Xms256m` sets the initial heap size to 256 MB.

- **-Xmx**

  This parameter controls the maximum size of the Java heap. Increasing this parameter increases the memory available to the application server, and reduces the frequency of garbage collection. Increasing this setting can improve server response time and throughput. However, increasing this setting also increases the duration of a garbage collection when it does occur. This setting should never be increased above the system memory available for the application server instance. Increasing the setting above the available system memory can cause system paging and a significant decrease in performance.

Information	Value
Default	By default, the JVM dynamically calculates the Java heap size based on the available memory in the system.
Recommended	Workload specific, but higher than the default value, depending on the amount of available physical memory.
Usage	Specifying `-Xmx512m` sets the maximum heap size to 512 MB.

Note: Specify a value for the **-Xmx** parameter to reduce possible out-of-memory issues.

4. Tune Java memory.

   Enterprise applications written in the Java language involve complex object relationships and use large numbers of objects. Although, the Java language automatically manages memory associated with object life cycles, understanding the application usage patterns for objects is important. In particular, verify that the following conditions exist:
   - The application is not over utilizing objects
   - The application is not leaking objects
   - The Java heap parameters are set properly to handle a given object usage pattern

   a. Check for over-utilization of objects.

      You can check if the application is overusing objects, by observing the counters for the JVM runtime. You have to specify the **-XrunpmiJvmtiProfiler** command-line option, as well as the JVM module maximum level in order to enable the Java virtual machine profiler interface, JVMTI, counters. The optimal result for the average time between garbage collections is at least five to six times the average duration of a single garbage collection. If you do not achieve this number, the application is spending more than 15 percent of its time in garbage collection.

      If the information indicates a garbage collection bottleneck, there are two ways to clear the bottleneck. The most cost-effective way to optimize the application is to implement object caches and pools. Use a Java profiler to determine which objects to target. If you can not optimize the application, try adding memory, processors and clones. Additional memory allows each clone to maintain a reasonable heap size. Additional processors allow the clones to run in parallel.

   b. Test for memory leaks.

      Memory leaks in the Java language are a dangerous contributor to garbage collection bottlenecks. Memory leaks are more damaging than memory overuse, because a memory leak ultimately leads to system instability. Over time, garbage collection occurs more frequently until the heap is exhausted and the Java code fails with a fatal out-of-memory exception. Memory leaks occur when an unused object has references that are never freed. Memory leaks most commonly occur in collection classes, such as Hashtable because the table always has a reference to the object, even after real references are deleted.

      High workload often causes applications to crash immediately after deployment in the production environment. If an application has memory leaks, a high workload can accelerate the magnification of the leakage and cause memory allocation failures to occur.

      The goal of memory leak testing is to magnify numbers. Memory leaks are measured in terms of the amount of bytes or kilobytes that cannot be garbage collected. The delicate task is to differentiate these amounts between expected sizes of useful and unusable memory. This task is

achieved more easily if the numbers are magnified, resulting in larger gaps and easier identification of inconsistencies. The following list provides insight on how to interpret the results of your memory leak testing:

- **Long-running test**

  Memory leak problems can manifest only after a period of time, therefore, memory leaks are found easily during long-running tests. Short running tests might provide invalid indications of where the memory leaks are occurring. It is sometimes difficult to know when a memory leak is occurring in the Java language, especially when memory usage has seemingly increased either abruptly or monotonically in a given period of time. The reason it is hard to detect a memory leak is that these kinds of increases can be valid or might be the intention of the developer. You can learn how to differentiate the delayed use of objects from completely unused objects by running applications for a longer period of time. Long-running application testing gives you higher confidence for whether the delayed use of objects is actually occurring.
- **Repetitive test**

  In many cases, memory leak problems occur by successive repetitions of the same test case. The goal of memory leak testing is to establish a big gap between unusable memory and used memory in terms of their relative sizes. By repeating the same scenario over and over again, the gap is multiplied in a very progressive way. This testing helps if the number of leaks caused by the execution of a test case is so minimal that it is hardly noticeable in one run.

  You can use repetitive tests at the system level or module level. The advantage with modular testing is better control. When a module is designed to keep the private module without creating external side effects such as memory usage, testing for memory leaks is easier. First, the memory usage before running the module is recorded. Then, a fixed set of test cases are run repeatedly. At the end of the test run, the current memory usage is recorded and checked for significant changes. Remember, garbage collection must be suggested when recording the actual memory usage by inserting System.gc() in the module where you want garbage collection to occur, or using a profiling tool, to force the event to occur.
- **Concurrency test**

  Some memory leak problems can occur only when there are several threads running in the application. Unfortunately, synchronization points are very susceptible to memory leaks because of the added complication in the program logic. Careless programming can lead to kept or not-released references. The incident of memory leaks is often facilitated or accelerated by increased concurrency in the system. The most common way to increase concurrency is to increase the number of clients in the test driver.

  Consider the following points when choosing which test cases to use for memory leak testing:
  - A good test case exercises areas of the application where objects are created. Most of the time, knowledge of the application is required. A description of the scenario can suggest creation of data spaces, such as adding a new record, creating an HTTP session, performing a transaction and searching a record.
  - Look at areas where collections of objects are used. Typically, memory leaks are composed of objects within the same class. Also, collection classes such as Vector and Hashtable are common places where references to objects are implicitly stored by calling corresponding insertion methods. For example, the get method of a Hashtable object does not remove its reference to the retrieved object.

Heap consumption that indicates a possible leak during periods when the application server is consistently near 100 percent CPU utilization, but disappears when the workload becomes lighter or near-idle, is an indication of heap fragmentation. Heap fragmentation can occur when the JVM can free sufficient objects to satisfy memory allocation requests during garbage collection cycles, but the JVM does not have the time to compact small free memory areas in the heap to larger contiguous spaces.

Another form of heap fragmentation occurs when objects that are less than 512 bytes are freed. The objects are freed, but the storage is not recovered, resulting in memory fragmentation until a heap compaction occurs.

Heap fragmentation can be reduced by forcing compactions to occur. However, there is a performance penalty for forcing compactions. Use the Java **-X** command to see the list of memory options.

5. Tune garbage collection

   The JVM uses a parallel garbage collector to fully exploit an SMP during most garbage collection cycles.

   Examining Java garbage collection gives insight to how the application is utilizing memory. Garbage collection is a Java strength. By taking the burden of memory management away from the application writer, Java applications are more robust than applications written in languages that do not provide garbage collection. This robustness applies as long as the application is not abusing objects. Garbage collection typically consumes from 5 to 20 percent of total run time of a properly functioning application. If not managed, garbage collection is one of the biggest bottlenecks for an application.

   Monitoring garbage collection while a fixed workload is running, provides you with insight as to whether the application is over using objects. Garbage collection can even detect the presence of memory leaks.

   You can use JVM settings to configure the type and behavior of garbage collection. When the JVM cannot allocate an object from the current heap because of lack of contiguous space, the garbage collector is invoked to reclaim memory from Java objects that are no longer being used. Each JVM vendor provides unique garbage collector policies and tuning parameters.

   You can use the **Verbose garbage collection** setting in the administrative console to enable garbage collection monitoring. The output from this setting includes class garbage collection statistics. The format of the generated report is not standardized between different JVMs or release levels.

   To adjust your JVM garbage collection settings:

   a. In the administrative console, click **Servers > Server Types > WebSphere application servers >** *server_name*.

   b. In the Server Infrastructure section, click **Java and process management > Process definition > servant**.

   c. Under additional properties, click **Environment entries**.

   d. Add or update the environment entry for IBM_JAVA_OPTIONS as follows.

      1) If you see an existing environment entry named IBM_JAVA_OPTIONS, edit it to append the –X Java option you want to add to the existing value.

      2) Otherwise, click **New** to create a new environment entry. Fill in the following values in their respective fields of the form:

Information	Value
Name:	IBM_JAVA_OPTIONS
Value:	The –X Java option you want to add
Description:	A description of that option

      This procedure updates the `was.env` file in the WebSphereApplication Server configuration directory. The change will apply the settings to all servant, control, and adjunct regions.

   e. Click **Apply**.

   f. Click **Save** to save your changes to the master configuration.

   g. Stop and restart the application server.

   The following list describes the –X options for the different JVM garbage collectors.

   The IBM virtual machine for Java garbage collector.

   A complete guide to the IBM implementation of the Java garbage collector is provided in the *IBM Developer Kit and Runtime Environment, Java2 Technology Edition, Version 5.0 Diagnostics Guide*. This document is available on the developerWorks® website.

   Use the Java **-X** option to view a list of memory options.

- **-Xgcpolicy**

  The IBM virtual machine for Java provides four policies for garbage collection. Each policy provides unique benefits.

  Note: While each policy provides unique benefits, for WebSphere Application Server Version 8.0 and later, gencon is the default garbage collection policy. Previous versions of the application server specify that optthruput is the default garbage collection policy.

  – gencon is the default policy. This policy works with the generational garbage collector. The generational scheme attempts to achieve high throughput along with reduced garbage collection pause times. To accomplish this goal, the heap is split into new and old segments. Long lived objects are promoted to the old space while short-lived objects are garbage collected quickly in the new space. The gencon policy provides significant benefits for many applications. However, it is not suited for all applications, and is typically more difficult to tune.

  – optthruput provides high throughput but with longer garbage collection pause times. During a garbage collection, all application threads are stopped for mark, sweep and compaction, when compaction is needed. The gencon policy is sufficient for most applications.

  – optavgpause is the policy that reduces garbage collection pause time by performing the mark and sweep phases of garbage collection while an application is running. This policy causes a small performance impact to overall throughput.

  – subpool is a policy that increases performance on multiprocessor systems, that commonly use more then 8 processors. This policy is only available on IBM System i® System p® and System z® processors. The subpool policy is similar to the gencon policy except that the heap is divided into subpools that provide improved scalability for object allocation.

Information	Value
Default	gencon
Recommended	gencon
Usage	Specifying `Xgcpolicy:gencon` sets the garbage collection policy to gencon.

Setting **gcpolicy** to `gencon` disables concurrent mark. You should get optimal throughput results when you use the gencon policy unless you are experiencing erratic application response times, which is an indication that you might have pause time problems

Setting **gcpolicy** to `optavgpause` enables concurrent mark with its default values. This setting alleviates erratic application response times that normal garbage collection causes. However, this option might decrease overall throughput.

- **-Xnoclassgc**

  By default, the JVM unloads a class from memory whenever there are no live instances of that class left. The overhead of loading and unloading the same class multiple times, can decrease performance.

  gotcha: You can use the `-Xnoclassgc` argument to disable class garbage collection. However, the performance impact of class garbage collection is typically minimal, and turning off class garbage collection in a Java Platform, Enterprise Edition (Java EE) based system, with its heavy use of application class loaders, might effectively create a memory leak of class data, and cause the JVM to throw an Out-of-Memory Exception.

If you use this option, whenever you redeploy an application, you should always restart the application server to clear the classes and static data from the pervious version of the application.

Information	Value
Default	Class garbage collection is enabled.
Recommended	Do not disable class garbage collection.
Usage	Specify `Xnoclassgc` to disable class garbage collection.

6. **Enable localhost name caching** By default in the IBM SDK for Java, the static method java/net/InetAddress.getLocalHost does not cache its result. This method is used throughout WebSphere Application Server, but particularly in administrative agents such as the deployment manager and node agent. If the localhost address of a process will not change while it is running, then it is advised to use a built-in cache for the localhost lookup by setting the com.ibm.cacheLocalHost system property to the value true. Refer to the Java virtual machine custom properties topic in the information center for instructions on setting JVM custom properties on the various types of processes.

   Note: The address for servers configured using DHCP change over time. Do not set this property unless you are using statically assigned IP addresses for your server.

Information	Value
Default	com.ibm.cacheLocalHost = *false*
Recommended	com.ibm.cacheLocalHost = *true* (see description)
Usage	Specifying `-Dcom.ibm.cacheLocalhost=true` enables the getLocalHost cache

7. **Enable class sharing in a cache.**

   The share classes option of the IBM implementation of the Java 2 Runtime Environment (J2RE) Version 1.5.0 lets you share classes in a cache. Sharing classes in a cache can improve startup time and reduce memory footprint. Processes, such as application servers, node agents, and deployment managers, can use the share classes option.

   If you use this option, you should clear the cache when the process is not in use. To clear the cache, either call the *app_server_root*`/bin/clearClassCache.bat/sh` utility or stop the process and then restart the process.

   If you need to disable the share classes option for a process, specify the generic JVM argument -Xshareclasses:none for that process:

   a. In the administrative console, click **Servers > Server Types > WebSphere application servers >** *server_name*.
   b. In the Server Infrastructure section, click **Java and process management > Process definition**
   c. Select either **Control** or **Servant**, and then select **Java virtual machine**.
   d. Enter `-Xshareclasses:none` in the **Generic JVM arguments** field.
   e. Click **OK**.
   f. Click **Save** to save your changes to the master configuration.
   g. Stop and restart the application server.

Information	Value
Default	The Share classes in a cache option are enabled.
Recommended	Leave the share classes in a cache option enabled.
Usage	Specifying `-Xshareclasses:none` disables the share classes in a cache option.

8. Enable compressed references on 64-bit environments.

   You can enable compressed references on 64-bit environments, such as AIX® 64, Linux PPC 64, zLinux 64, and Microsoft Windows AMD64, Linux AMD64.

   The compressed references option of the IBM implementation of the 64-bit Java SE Runtime Environment (JRE) Version 6.0 lets you limit all of the memory references to 32-bit size. Typically, the 64-bit JVMs use more heap space than the 32-bit JVMs because they use 64-bit wide memory references to address memory. The heap that is addressable by the 64-bit reference is orders of magnitude larger than the 32-bit heap, but in the real world, a heap that requires all 64-bits for addressing is typically not required. Compressing the references reduces the size of the addresses and makes more efficient use of the heap. Compressing these references also improves the processor cache and bus utilization, thereby improving performance.

   gotcha:

   The compressed references feature is not supported on:

   - HP-UX 64-bit JVM
   - iSeries® Classic 64-bit JVM

   To enable a 64-bit JVM to run in the compressed references mode, you need to specify a new environment variable in WebSphereApplication Server configuration. In the administrative console:

   a. Click: **Servers > Server Types > WebSphere application servers > server_name**.
   b. Click the Configuration tab. Under Server Infrastructure, click **Java and process management > ProcessDefinition > servant**.
   c. Under additional properties, click **Environment entries**.

      Add or update the environment entry for IBM_JAVA_OPTIONS as follows:

      1) If you see an existing environment entry named IBM_JAVA_OPTIONS, edit it to append the Java option –Xcompressedrefs to the existing value.
      2) Otherwise, click **New** to create a new environment entry. Fill in following values in their respective fields of the form:

Information	Value
Name:	IBM_JAVA_OPTIONS
Value:	-Xcompressedrefs
Description:	Enable 64-bit compressed references mode

   This procedure updates the ‘was.env' file in the WebSphereApplication Server configuration directory. The change will apply the settings to all servant, control, and adjunct regions.

   gotcha: When you supply Xcompressedrefs as a generic JVM argument, WebSphereApplication Server will fail to start because of an unsupported Java option error. If the application requires more than a 30 GB Java heap, then 64-bit default mode should be used

   The product automatically enables pointer compression on the supported platforms by default if the heap size (controlled by the -Xmx parameter) is set under a certain heap size (around 25 GB depending on platform), else it will default to non-compressed references. The user can override these defaults by using the command line options below.

   The following command-line options control compressed references feature:

   -Xcompressedrefs

   This command-line option enables the compressed references feature. When the JVM is launched with this command line option it would use 32-bit wide memory references to address the heap. This feature can be used up to a certain heap size (around 29GB depending on the platform), controlled by -Xmx parameter.

   -Xnocompressedrefs

   This command-line options explicitly disable the compressed references feature. When the

JVM is launches with this command line option it will use full 64-bit wide memory references to address the heap. This option can be used by the user to override the default enablement of pointer compression, if needed.

9. Tune the configuration update process for a large cell configuration.

   In a large cell configuration, you might need to determine whether configuration update performance or consistency checking is more important. The deployment manager maintains a master configuration repository for the entire cell. By default, when the configuration changes, the product compares the configuration in the workspace with the master repository to maintain workspace consistency. However, the consistency verification process can cause an increase in the amount of time to save a configuration change or to deploy a large number of applications. The following factors influence how much time is required:

   - The more application servers or clusters that are defined in a cell, the longer it takes to save a configuration change.
   - The more applications that are deployed in a cell, the longer it takes to save a configuration change.

   If the amount of time required to change a configuration change is unsatisfactory, you can add the `config_consistency_check` custom property to your JVM settings and set the value of this property to false.

   a. In the administrative console, click **System administration > Deployment manager**.
   b. Under Server Infrastructure, select Java and Process Management, and then click **Process Definition**.
   c. Under Additional Properties, click **Java Virtual Machine > Custom Properties > New**.
   d. Enter `config_consistency_check` in the Name field and `false` in the Value field.
   e. Click **OK** and then save these changes to the master configuration.
   f. Restart the server.

   Note: The `config_consistency_check` custom property affects the deployment manager process only. It does not affect other processes including the node agent and application server processes. The consistency check is not performed on these processes. However, within the `SystemOut.log` files for these processes, you might see a note that the consistency check is disabled. For these non-deployment manager processes, you can ignore this message.

   If you are using the wsadmin command **`wsadmin -conntype none`** in local mode, you must set the **config_consistency_check** property to `false` before issuing this command.

What to do next

Continue to gather and analyze data as you make tuning changes until you are satisfied with how the JVM is performing.

Directory conventions

References in product information to *app_server_root*, *profile_root*, and other directories imply specific default directory locations. This article describes the conventions in use for WebSphere Application Server.

Default product locations - z/OS

app_server_root

Refers to the top directory for a WebSphere Application Server node.

The node may be of any type—application server, deployment manager, or unmanaged for example. Each node has its own *app_server_root*. Corresponding product variables are *was.install.root* and *WAS_HOME*.

The default varies based on node type. Common defaults are *`configuration_root`*`/AppServer` and *`configuration_root`*`/DeploymentManager`.

configuration_root
Refers to the mount point for the configuration file system (formerly, the configuration HFS) in WebSphere Application Server for z/OS.

The *configuration_root* contains the various *app_server_root* directories and certain symbolic links associated with them. Each different node type under the *configuration_root* requires its own cataloged procedures under z/OS.

The default is `/wasv8config/cell_name/node_name`.

plug-ins_root
Refers to the installation root directory for Web Server Plug-ins.

profile_root
Refers to the home directory for a particular instantiated WebSphere Application Server profile.

Corresponding product variables are *server.root* and *user.install.root.*

In general, this is the same as `app_server_root/profiles/profile_name`. On z/OS, this will always be `app_server_root/profiles/default` because only the profile name "default" is used in WebSphere Application Server for z/OS.

smpe_root
Refers to the root directory for product code installed with SMP/E or IBM Installation Manager.

The corresponding product variable is *smpe.install.root*.

The default is `/usr/lpp/zWebSphere/V8R5`.

Tuning transport channel services

The transport channel services manage client connections and I/O processing for HTTP and JMS requests. These I/O services are based on the non-blocking I/O (NIO) features that are available in Java. These services provide a highly scalable foundation to WebSphere Application Server request processing. Java NIO-based architecture has limitations in terms of performance, scalability, and user usability. Therefore, integration of true asynchronous I/O is implemented. This implementation provides significant benefits in usability, reduces the complexity of I/O processing, and reduces that amount of performance tuning you must perform.

About this task

Key features of the new transport channel services include:

- Scalability, which enables the product to handle many concurrent requests
- Asynchronous request processing, which provides a many-to-one mapping of client requests to web container threads
- Resource sharing and segregation, which enables thread pools to be shared between the web container and a messaging service
- Improved usability
- Incorporation of autonomic tuning and configuration functions

Changing the default values for settings on one or more of the transport channels associated with a transport chain can improve the performance of that chain.

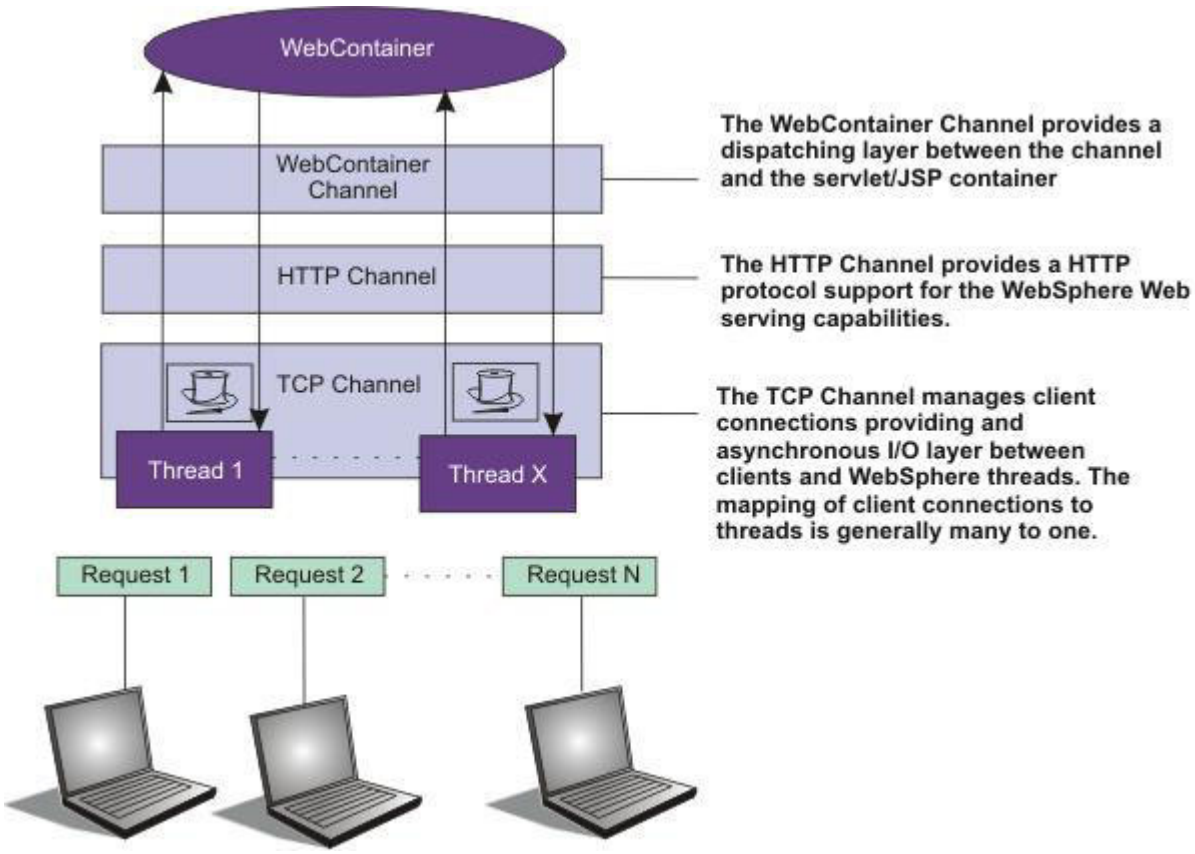


Figure 1. Transport Channel Service

Procedure

- Adjust TCP transport channel settings. In the administration console, click **Servers > Server Types > WebSphere application servers >** *server_name* **> Ports**. Then click **View associated transports** for the appropriate port.
  1. Select the transport chain whose properties you are changing.
  2. Click the TCP transport channel defined for that chain.
  3. Lower the value specified for the Maximum open connections property. This parameter controls the maximum number of connections that are available for a server to use. Leaving this parameter at the default value of `20000`, which is the maximum number of connections, might lead to stalled websites under failure conditions, because the product continues to accept connections, thereby increasing the connection, and associated work, backlog. The default should be changed to a significantly lower number, such as `500`, and then additional tuning and testing should be performed to determine the optimal value that you should specify for a specific website or application deployment.
  4. If client connections are being closed without data being written back to the client, change the value specified for the Inactivity timeout parameter. This parameter controls the maximum number of connections available for a server use. After receiving a new connection, the TCP transport channel waits for enough data to arrive to dispatch the connection to the protocol-specific channels above the TCP transport channel. If not enough data is received during the time period specified for the Inactivity timeout parameter, the TCP transport channel closes the connection.

     The default value for this parameter is `60` seconds. This value is adequate for most applications. Increase the value specified for this parameter if your workload involves many connections and all of these connections cannot be serviced in 60 seconds.

- Adjust HTTP transport channel settings. In the administration console, click **Servers > Server Types > WebSphere application servers >** *server_name* **> Ports**. Then click **View associated transports** for the appropriate port.
  1. Select the transport chain whose properties you are changing.
  2. Click the HTTP transport channel defined for that chain.
  3. Tune HTTP keep-alive.

     The Use persistent (keep-alive) connections setting controls whether connections are left open between requests. Leaving the connections open can save setup and teardown costs of sockets if your workload has clients that send multiple requests. The default value is true, which is typically the optimal setting.

     If your clients only send single requests over substantially long periods of time, it is probably better to disable this option and close the connections right away rather than to have the HTTP transport channel setup the timeouts to close the connection at some later time.
  4. Change the value specified for the Maximum persistent requests parameter to increase the number of requests that can flow over a connection before it is closed.

     When the **Use persistent connections** option is enabled, the Maximum persistent requests parameter controls the number of requests that can flow over a connection before it is closed. The default value is `100`. This value should be set to a value such that most, if not all, clients always have an open connection when they make multiple requests during the same session. A proper setting for this parameter helps to eliminate unnecessary setting up and tearing down of sockets.

     For test scenarios in which the client is never closed, a value of `-1` disables the processing which limits the number of requests over a single connection. The persistent timeout shuts down some idle sockets and protects your server from running out of open sockets.
  5. Change the value specified for the Persistent timeout parameter to increase the length of time that a connection is held open before being closed due to inactivity. The Persistent timeout parameter controls the length of time that a connection is held open before being closed because there is no activity on that connection. The default value is `30` seconds This parameter is set to a value that keeps enough connections open so that most clients can obtain a connection available when they must make a request.
  6. If clients are having trouble completing a request because it takes them more than `60` seconds to send their data, change the value specified for the Read timeout parameter. Some clients pause more than `60` seconds while sending data as part of a request. To ensure that they are able to complete their requests, change the value specified for this parameter to a length of time in seconds that is sufficient for the clients to complete the transfer of data. Be careful when changing this value that you still protect the server from clients who send incomplete data and thereby use resources (sockets) for an excessive amount of time.
  7. If some of your clients require more than `60` seconds to receive data being written to them, change the value specified for the Write timeout parameter. Some clients are slow and require more than `60` seconds to receive data that is sent to them. To ensure that they are able to obtain all of their data, change the value specified for this parameter to a length of time in seconds that is sufficient for all of the data to be received. Be careful when changing this value that you still protect the server from malicious clients.
- Adjust the web container transport channel settings. In the administration console, click **Servers > Server Types > WebSphere application servers >** *server_name* **> Ports**. Then click **View associated transports** for the appropriate port.
  1. Select the transport chain whose properties must be changed.
  2. Click the web container transport channel defined for that chain.
  3. If multiple writes are required to handle responses to the client, change the value specified for the Write buffer size parameter to a value that is more appropriate for your clients. The Write buffer size parameter controls the maximum amount of data per thread that the web container buffers before sending the request on for processing. The default value is `32768` bytes, which are sufficient for

most applications. If the size of a response is greater than the size of the write buffer, the response is chunked and written back in multiple TCP writes.

If you must change the value specified for this parameter, make sure that the new value enables most requests to be written out in a single write. To determine an appropriate value for this parameter, look at the size of the pages that are returned and add some additional bytes to account for the HTTP headers.

- **Adjust the settings for the bounded buffer.**

  Even though the default bounded buffer parameters are optimal for most of the environments, you might want to change the default values in certain situations and for some operating systems to enhance performance. Changing the bounded buffer parameters can degrade performance. Therefore, make sure that you tune the other related areas, such as the web container and ORB thread pools, before deciding to change the bounded buffer parameters.

  To change the bounded buffer parameters:

  1. In the administrative console, click **Servers > Server Types > WebSphere application servers >** *server_name*.
  2. In the Server Infrastructure section, click **Java and process management > Process definition > Java virtual machine**.
  3. Click **Custom properties**.
  4. Enter one of the following custom properties in the Name field and an appropriate value in the **Value** field, and then click **Apply** to save the custom property and its setting.
     - com.ibm.ws.util.BoundedBuffer.spins_take=*value*

       Specifies the number of times a web container thread can attempt to retrieve a request from the buffer before the thread is suspended and enqueued. This parameter enables you to trade off the cost of performing possibly unsuccessful retrieval attempts, with the cost to suspending a thread and activating it again in response to a put operation.

Information	Value
Default:	The number of processors available to the operating system minus 1.
Recommended:	Use any non-negative integer value. In practice, using an integer from 2 to 8 yields the best performance results.
Usage:	com.ibm.ws.util.BoundedBuffer.spins_take=6. Six attempts are made before the thread is suspended.

     - com.ibm.ws.util.BoundedBuffer.yield_take=true or false

       Specifies that a thread yields the processor to other threads after a set number of attempts to take a request from the buffer. Typically a lower number of attempts is preferable.

Information	Value
Default:	false
Recommended:	The effect of yield is implementation-specific for individual platforms.
Usage:	com.ibm.ws.util.BoundedBuffer.spins_take=*boolean value*

     - com.ibm.ws.util.BoundedBuffer.spins_put=*value*

       Specifies the number of attempts an InboundReader thread makes to put a request into the buffer before the thread is suspended and enqueued. Use this value to trade off between the cost of repeated, possibly unsuccessful, attempts to put a request into the buffer with the cost to suspend a thread and reactivate it in response to a take operation.

Information	Value
Default:	The value of com.ibm.ws.util.BoundedBuffer.spins_take divided by 4.
Recommended:	Use any non-negative integer value. In practice an integer 2 - 8 have shown the best performance results.
Usage:	com.ibm.ws.util.BoundedBuffer.spins_put=6. Six attempts are made before the thread is suspended.

– com.ibm.ws.util.BoundedBuffer.yield_put=true or false

Specifies that a thread yields the processor to other threads after a set number of attempts to put a request into the buffer. Typically a lower number of attempts is preferable.

Information	Value
Default:	false
Recommended:	The effect of yield is implementation-specific for individual platforms.
Usage:	com.ibm.ws.util.BoundedBuffer.yield_put=*boolean value*

– com.ibm.ws.util.BoundedBuffer.wait=*number of milliseconds*

Specifies the maximum length of time, in milliseconds, that a request might unnecessarily be delayed if the buffer is completely full or if the buffer is empty.

Information	Value
Default:	10000 milliseconds
Recommended:	A value of 10000 milliseconds usually works well. In rare instances when the buffer becomes either full or empty, a smaller value guarantee a more timely handling of requests, but there is usually a performance impact to using a smaller value.
Usage:	com.ibm.ws.util.BoundedBuffer.wait=8000. A request might unnecessarily be delayed up to 8000 milliseconds.

- Click **Apply** and then click **Save** to save these changes.

Checking hardware configuration and settings

An optimal hardware configuration enables applications to get the greatest benefit from performance tuning. The hardware speed impacts all types of applications and is critical to overall performance.

About this task

You can check hardware configuration and settings such as disk speed, system memory and processor speed to gain performance benefits.

Procedure

- Use the following considerations for selecting and configuring the hardware on which the application servers run:
  1. Optimize disk speed
     – Description: Disk speed and configuration have a dramatic effect on the performance of application servers running applications that are heavily dependent on the database support, using extensive messaging, or processing workflow. The disk input or output subsystems that are

optimized for performance, for example Redundant Array of Independent Disks (RAID) array, high-speed drives, and dedicated caches, are essential components for optimum application server performance in these environments.

Application servers with fewer disk requirements can benefit from a mirrored disk drive configuration that improves reliability and has good performance.

– Recommendation: Spread the disk processing across as many disks as possible to avoid contention issues that typically occur with 1- or 2-disk systems. Placing the database tables on disks that are separate from the disks that are used for the database log files reduces disk contention and improve throughput.

2. Increase processor speed and processor cache
   – **Description:** In the absence of other bottlenecks, increasing the processor speed often helps throughput and response times. A processor with a larger L2 or L3 cache yields higher throughput, even if the processor speed is the same as a CPU with a smaller L2 or L3 cache.
3. Increase system memory
   – Description: Increase memory to prevent the system from paging memory to the disk to improve performance. Allow a minimum of 256 MB of memory for each processor and 512 MB per application server. Adjust the available memory when the system pages and the processor utilization is low because of the paging. The memory access speed might depend on the number and placement of the memory modules. Check the hardware manual to make sure that your configuration is optimal.
   – Recommendation: Use 256 MB of memory for each processor and 512 MB per application server. Some applications might require more memory.
   – Description: The amount of storage required for z/OS is mostly dependent on the number of servers and the size of the Java Virtual Machine (JVM) heap for each server.
   – Recommendation: For a single server with 1 GB JVM heap, allocate a minimum of 1GB of memory.
4. Increase system memory
   – Description: Increase memory to prevent the system from paging memory to the disk to improve performance. Allow a minimum of 256 MB of memory for each processor and 512 MB per application server. Adjust the available memory when the system pages and the processor utilization is low because of the paging. The memory access speed might depend on the number and placement of the memory modules. Check the hardware manual to make sure that your configuration is optimal.
   – Recommendation: Use 256 MB of memory for each processor and 512 MB per application server. Some applications might require more memory.
   – Description: The amount of storage required for z/OS is mostly dependent on the number of servers and the size of the Java Virtual Machine (JVM) heap for each server.
   – Recommendation: For a single server with 1 GB JVM heap, allocate a minimum of 1GB of memory.
5. Run network cards and network switches at full duplex
   – Description: Run network cards and network switches at full duplex and use the highest supported speed. Full duplex is much faster than half duplex. Verify that the network speed of adapters, cables, switches, and other devices can accommodate the required throughput. Some websites might require multiple gigabit links.
   – Recommendation Make sure that the highest speed is in use on 10/100/1000 Ethernet networks.
6. An **IBM S/390® or zSeries® Model** that supports the software requirement of z/OS V1R2.
7. Storage
   – Storage requirements are higher than for traditional workloads
   – Recommendation
     - Virtual storage default should be about 370 MB per servant, which includes a 256 MB default heap size and a default initial LE heap size of 80 MB.
     -

Note: Real storage minimum is 512 MB per LPAR for a light load such as the IVP. For most real-world applications, you should use 2 GB or higher. We have seen applications that require as much as 8 GB of real to operate at peak load.

- DASD
  - Recommendation
    -

      Note: To maximize your performance, you should use a fast DASD subsystem (for example, IBM Shark), running with a high cache read/write hit rate.
- Networking
  - Recommendation
    -

      Note: For high bandwidth applications, you should use at least a 1 Gb Ethernet connection. If your applications have extremely high bandwidth requirements, you may need additional Ethernet connections.
- Cryptography
  - Recommendation
    -

      Note: For applications that make heavy use of cryptography, you should use the zSeries or S/390 cryptographic hardware and the Integrated Cryptographic Service Facility. For more information, refer to the zSeries and S/390 Cryptography website.

Tuning operating systems

You can tune your operating system to optimize the performance of WebSphere Application Server.

About this task

Tuning parameters are specific to operating systems. Because these operating systems are not WebSphere Application Server products, be aware that the products can change and results can vary.

Note: Check your operating system documentation to determine how to make the tuning parameters changes permanent and if a reboot is required.

Procedure

Tune z/OS operating systems

Tuning on z/OS

This page provides a starting point to tune your z/OS operating system to optimize WebSphere Application Server performance.

About this task

One of the goals of the WebSphere Application Server for z/OS programming model and runtime environment is to significantly simplify the work required for application developers to write and deploy applications. WebSphere Application Server for z/OS can help to relieve the application programmer of many of the plumbing tasks involved in developing applications. For example, application code in WebSphere Application Server for z/OS does not concern itself directly with remote communication, it locates objects which might be local or remote, and drives methods. As a result, there is no direct use of socket calls or TCP/IP programming in a WebSphere Application Server for z/OS application.

Separating application development tasks from determining where to run the application is one aspect of relieving plumbing tasks. Other aspects to consider include assistance with data calls for multiple types of beans, assistance with user authentication, and threading.

Removing tuning work from the application programmer does not mean that the work is complete. There might also be tuning work for database administrators, network administrators, security administrators, and performance analysts.

Use the following topics to tune your z/OS operating system to optimize WebSphere Application Server performance.

Procedure

- Tune the z/OS operating system
- Tune the WebSphere Application Server for z/OS runtime
- Tune the message-driven bean processing on z/OS by using WebSphere MQ as the messaging provider in ASF mode

What to do next

For more information about tuning applications, see the using application clients information.

Tuning the z/OS operating system

Use these steps to tune your z/OS operating system to optimize WebSphere Application Server performance.

Procedure

1. Tuning storage
2. “z/OS operating system tuning tips”
3. “UNIX System Services (USS) tuning tips for z/OS” on page 54
4. “Workload management (WLM) tuning tips for z/OS” on page 59
5. “Resource Recovery Service (RRS) tuning tips for z/OS” on page 51
6. “Fine tuning the LE heap” on page 55

z/OS operating system tuning tips

There are several configuration changes you can make to z/OS system components that might improve product performance.

You might want to make one or more of the following changes to the indicated z/OS components:

- CTRACE

  The first place to review is your CTRACE configuration. Ensure that all components are either set to MIN or OFF. To display the CTRACE options for all components on your system, issue the following command from the operator console:

  ```
  D TRACE,COMP=ALL
  ```

  To change the setting for an individual component to its minimum tracing value, use the following command, where xxx is the component ID.

  ```
  TRACE CT,OFF,COMP=xxx
  ```

  This configuration change eliminates the unnecessary overhead of collecting trace information that is not needed. Often during debug, CTRACE is turned on for a component and not shut off when the problem is resolved.
- SMF

Ensure that you are not collecting more SMF data than you need. Review the SMFPRM*xx* settings to ensure that only the minimum number of records are collected.

Use SMF 92 or 120 only for diagnostics.

– SMF Type 92

SMF Type 92 records are created each time an HFS file is opened, closed, deleted, and so forth. Almost every web server request references HFS files, so thousands of SMF Type 92 records are created. Unless you specifically need this information, turn off SMF Type 92 records. In the following example, we have disabled the collection of SMF type 92 records:

Example:

```
ACTIVE,
DSNAME(SYS1.&.SYSNAME..SMF.MAN1;SYS1.&SYSNAME..SMF.MAN2;),
NOPROMPT,
REC(PERM),
MAXDORM(3000),
STATUS(010000),
JWT(0510),
SID(&SYSNAME;(1:4)),
LISTDSN,
SYS(NOTYPE(19,40,92)),
INTVAL(30),
SYNCVAL(00),
SYS(DETAIL,INTERVAL(SMF,SYNC)),
SYS(EXITS(IEFACTRT,IEFUJI,IEFU29,IEFU83,IEFU84,IEFU85,IEFUJV,IEFUSI))
```

– SMF Type 120

Note:

You might find that running with SMF 120 records in production is appropriate, because these records give information specific to applications that are running on the product, such as response time for Enterprise Edition (Java EE) artifacts, bytes transferred, and so forth. If you do choose to run with SMF 120 records enabled, you should use server interval SMF records and container interval SMF records rather than server activity records and container activity records. See the *Troubleshooting and support* PDF for a description of the SMF 120 record.

The *Troubleshooting and support* PDF also describes the steps involved in controlling collection of SMF 120 records. To enable specific record types, specify the following properties:

- server_SMF_server_activity_enabled=0 (or server_SMF_server_activity_enabled = false)
- server_SMF_server_interval_enabled=1 (or server_SMF_server_interval_enabled = true)
- server_SMF_container_activity_enabled=0 (or false)
- server_SMF_container_interval_enabled=1 (or true)
- server_SMF_interval_length=1800
- server_SMF_request_activity_enabled=1 (or true)
- server_SMF_outbound_enabled=1 (or true)

- You might also want to review your DB2 records and the standard RMF™ written SMF records, and ensure that the SMF data sets are allocated optimally. DB2 SMF records 100, 101 and 102 affect performance and should be used only for monitoring DB2 performance with the DB2 PM tool. If you are not monitoring DB2 performance, you should consider not collecting those SMF records.

Resource Recovery Service (RRS) tuning tips for z/OS

Use these tips to tune your z/OS operating system to optimize WebSphere Application Server performance.

- For best throughput, use coupling facility (CF) logger for the RRS log.

  DASD logger can limit your throughput because it is I/O-sensitive. The CF logger has much more throughput (in one measurement, the CF logger was six times faster than the DASD logger). Throughput will benefit from moving the RRS logs in logger to a coupling facility (CF) logstream. Doing so will help transactions complete quickly and not require any DASD I/O. If it's not possible to use CF logs, use well performing DASD and make sure the logs are allocated with large CI sizes.

- Ensure that your CF logger configuration is optimal by using SMF 88 records.

  See the tuning section of *z/OS MVS™ Setting Up a Sysplex* or the chapter on System Logger Accounting in *z/OS MVS System Management Facilities (SMF)* for details. In any case, you should monitor the logger to ensure that there is a sufficient size in the CF and that offloading is not impacting the overall throughput. The transaction logs are shared I/O-intensive resources in the mainline and can affect throughput dramatically if mistuned.
- Set adequate default values for the LOGR policy.

  bprac: Default values of LOGR policy may have an impact on performance. You might want to use the default settings in the table below.

Table 3. Recommended default setting for LOGR

Log Stream	Initial Size	Size
RM.DATA	1 MB	1MB
MAIN.UR	5 MB	50 MB
DELAYED .UR	5 MB	50 MB
RESTART	1 MB	5 MB
ARCHIVE	5 MB	50 MB

- Review XA Resource Managers log sizes.

  If you are using XA Resource Managers and you have chosen to put the logs in the logger, you may have to review the log sizes. As of this writing, we cannot give specific recommendations.

  Note: You can configure the XA logs in the install dialog to live either in the HFS or in logstreams. If you are not using global transactions involving XA resources, there is no point in putting the log in a logstream. If the XA logs are placed in logstreams, they should be in the Coupling Facility instead of DASD. The default names are 'HLQ.server.M' and 'HLQ.server.D' where HLQ is a user-defined value between 1-8 characters specified in the install dialog, and 'server' is the server short name. It is the installer's responsibility to ensure that the HLQ + server name is unique across the configuration. If it is not, the server will fail to start because the user data in the existing logstream will not match that of the new server. The logs (and structures, if applicable) are created in job 'BBOLOGSA' in the install dialog. If structures need to be allocated, there is also a step indicating what structure names need to be added to the CFRM policy. 5MB initial and 20MB max sizes should be used for both of these logstreams.

  Note: Set AUTODELETE(NO) for all logstreams.
- Eliminate archive log if not needed.

  Note: If you don't need the archive log, you should eliminate it since it can introduce extra DASD I/Os. The archive log contains the results of completed transactions. Normally, the archive log is not needed. Following is an example of disabling archive logging.

  Example:

```
//STEP1 EXEC PGM=IXCMIAPU
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
    DATA TYPE(LOGR)
    DELETE LOGSTREAM NAME(ATR.WITPLEX.ARCHIVE)

    DELETE LOGSTREAM NAME(ATR.WITPLEX.MAIN.UR)
    DELETE LOGSTREAM NAME(ATR.WITPLEX.RESTART)
    DELETE LOGSTREAM NAME(ATR.WITPLEX.RM.DATA)
    DELETE LOGSTREAM NAME(ATR.WITPLEX.DELAYED.UR)
    DELETE STRUCTURE NAME(RRSSTRUCT1)
/*
```

```
//STEP2 EXEC PGM=IXCMIAPU
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
    DATA TYPE(LOGR)
    DEFINE STRUCTURE NAME(RRSSTRUCT1)
           LOGSNUM(9)


    DEFINE LOGSTREAM NAME(ATR.WITPLEX.MAIN.UR)
           STRUCTNAME(RRSSTRUCT1)
           STG_DUPLEX(YES)
           DUPLEXMODE(UNCOND)
           LS_DATACLAS(SYSPLEX)
           LS_STORCLAS(LOGGER)
           HLQ(IXGLOGR)
           AUTODELETE(NO)
           RETPD(3)
    DEFINE LOGSTREAM NAME(ATR.WITPLEX.RESTART)
           STRUCTNAME(RRSSTRUCT1)
           STG_DUPLEX(YES)
           DUPLEXMODE(UNCOND)
           LS_DATACLAS(SYSPLEX)
           LS_STORCLAS(LOGGER)
           HLQ(IXGLOGR)
           AUTODELETE(NO)
           RETPD(3)
    DEFINE LOGSTREAM NAME(ATR.WITPLEX.RM.DATA)
           STRUCTNAME(RRSSTRUCT1)
           STG_DUPLEX(YES)
           DUPLEXMODE(UNCOND)
           LS_DATACLAS(SYSPLEX)
           LS_STORCLAS(LOGGER)
           HLQ(IXGLOGR)
           AUTODELETE(NO)
           RETPD(3)
    DEFINE LOGSTREAM NAME(ATR.WITPLEX.DELAYED.UR)
           STRUCTNAME(RRSSTRUCT1)
           STG_DUPLEX(YES)
           DUPLEXMODE(UNCOND)
           LS_DATACLAS(SYSPLEX)
           LS_STORCLAS(LOGGER)
           HLQ(IXGLOGR)
           AUTODELETE(NO)
           RETPD(3)
/*

//* DEFINE LOGSTREAM NAME(ATR.WITPLEX.ARCHIVE)
           //* STRUCTNAME(RRSSTRUCT1)
           //* STG_DUPLEX(YES)
           //* DUPLEXMODE(UNCOND)
           //* LS_DATACLAS(SYSPLEX)
           //* LS_STORCLAS(LOGGER)
           //* HLQ(IXGLOGR)
           //* AUTODELETE(NO)
           //* RETPD(3)
```

Note: RRS manual recommends creating one structure per log stream.

LE tuning tips for z/OS

Enabling xplink in the runtime environment and compiling applications with xplink enabled improves performance in z/OS V1R2.

- For best performance, use the LPALSTxx parmlib member to ensure that LE and C++ runtimes are loaded into LPA, as shown in the following example:

  Example: sys1.parmlib(LPALSTxx):

```
******************************** Top of Data ********************
USER.LPALIB,
ISF.SISFLPA,                          SDSF
CEE.SCEELPA,
                          LANGUAGE ENVIRONMENT
CBC.SCLBDLL,                          C++ RUNTIME
.
.
.

******************************** Bottom of Data ******************
```

- Ensure that the Language Environment® data sets, SCEERUN and SCEERUN2, are authorized to enable xplink.

  For processes that run the client ORB, since they start the JVM, must run with xplink (on). For best performance, compile applications that use JNI services with xplink enabled. Compiling applications with xplink enabled improves performance in z/OS V1R2. As you move from z/OS V1R2 to z/OS V1R6 you should experience additional performance improvements when all of the LE services calls are xplink enabled.
- Ensure that you are **NOT** using the following options during production:
  - RPTSTG(ON)
  - RPTOPTS(ON)
  - HEAPCHK(ON)
- Turn LE heappools on.

  If you are running a client on z/OS, setting the following: SET LEPARM='HEAPP(ON)' in a shell script, turns on LE heappools, which should improve the performance of the client.
- Refer to “Fine tuning the LE heap” on page 55

Customization Note: Do not modify LE parameters without consulting IBM support. The LE parameters are set internally to ensure the best possible performance of the WebSphere Application Server, which is the main LE application running in the address space. If you need to add or change LE parameters, make sure that you work with the IBM WebSphere support team to ensure that the internally set parameters are not compromised. The appropriate interface for making these changes is through the `PARM=` parameter of the `EXEC PGM=BPXBATSL` statement in the startup JCL.

UNIX System Services (USS) tuning tips for z/OS

Use these tips to tune your z/OS operating system to optimize WebSphere Application Server performance.

WebSphere Application Server for z/OS no longer requires or recommends the shared file system for the configuration files, since it maintains its own mechanism for managing this data in a cluster. However, WebSphere for z/OS does require the shared files system for XA partner logs. Your application may also use the shared file system. This article provides some basic tuning information for the shared file system.

For basic z/OS UNIX System Services performance information, refer to the following website: http://www.ibm.com/servers/eserver/zseries/ebusiness/perform.html

- Consider using zFS.

  z/OS has introduced a new file system called zFS which should provide improved file system access. You may benefit from using the zFS for your UNIX file system. See *z/OS UNIX System Services Planning* for more information.
- Decide whether to mount the shared file system R/W or R/O.

Starting with z/OS Version 1.13, if zFS is used as the file system in a sysplex, all of the systems in that sysplex have direct access to that file system. In this situation, you can mount the WebSphere Application Server file system R/W in a shared file system environment without negatively impacting performance.

If you are running in a sysplex environment on z/OS Version 1.12 or earlier, or if zFS is not used as the file system for your sysplex, you must give special consideration to how you mount the WebSphere Application Server file system. When a file system is mounted R/W in a shared file system environment on these operating systems, only one system has local access to the files. All other systems have remote access to the files which negatively affects performance. Therefore, you might want to put all of the files for WebSphere in their own mountable file system and mount it R/O to improve performance. However, to change your current application or install new applications, the file system must be mounted R/W. You will need to put operational procedures in place to ensure that the file system is mounted R/W when updating or installing applications.

- Determine which files are good candidates for HFS files caching.

  HFS Files Caching Read/Write files are cached in kernel data spaces. In order to determine what files would be good candidates for file caching you can use SMF 92 records.

  Initial cache size is defined in BPXPRMxx.

- Use the filecache command.

  High activity, read-only files can be cached in the USS kernel using the filecache command. Access to files in the filecache can be much more efficient than access to files in the shared file system, even if the shared file system files are cached in dataspaces. GRS latch contention, which sometimes is an issue for frequently accessed files shared file system, will not affect files in the filecache.

  To filecache important files at startup, you can add filecache command to your /etc/rc file. Unfortunately, files which are modified after being added to the filecache may not be eligible for caching until the file system is unmounted and remounted, or until the system is re-IPLed. Refer to *z/OS UNIX System Services Command Reference* for more information about the filecache command.

  Example of using the filecache command:

  ```
  /usr/sbin/filecache -a /usr/lpp/WebSphere/V5R0M0/
  MQSeries/java/samples/base/de_DE/mqsample.html
  ```

- Do not enable global audit ALWAYS on the RACF (SAF) classes that control access to objects in the UNIX file system. If audit ALWAYS is specified in the SETR LOGOPTIONS for DIRACC, DIRSRCH, FSOBJ or FSSEC very severe performance degradation might occur. If auditing is required, audit only failures using SETR LOGOPTIONS, and audit successes for only selected objects that require it. After enabling any auditing on these classes, verify that the change did not cause an unacceptable impact on response times and CPU usage.

Fine tuning the LE heap

Use these steps to tune your z/OS operating system to optimize WebSphere Application Server performance.

About this task

The LE Heap is an area of storage management to be concerned with. For servers, IBM has compiled default values for HEAP and HEAPPOOL into the server main programs. These are good starting points for simple applications. To fine tune the LE Heap settings, use the following procedure:

Procedure

1. Use the LE RPTSTG(ON) function to generate a report on storage utilization for your application servers. Perform the following actions to enable this function. The results appear in the servant joblog.
   a. In the administrative console, click **Environment** > **WebSphere variables >** > **New**.
   b. Specify `_CEE_RUNOPTS` in the **Name** field, and `RPTSTG(ON),RPTOPTS(ON)` in the **Value** field.
   c. Click **Save** to save your changes
2. To bring the server down cleanly, use the following VARY command:

```
VARY WLM,APPLENV=xxxx,QUIESCE
```

The following example shows the servant SYSPRINT DD output from the RPTSTG(ON) function.

Example:

```
.   .   .
0    HEAP statistics:
       Initial size:                                  83886080

       Increment size:                                 5242880
       Total heap storage used (sugg. initial size):  184809328

       Successful Get Heap requests:                    426551
       Successful Free Heap requests:                   424262
       Number of segments allocated:                         1
       Number of segments freed:                             0
   .   .   .

  Suggested Percentages for current Cell Sizes:
    HEAPP(ON,8,6,16,4,80,42,808,45,960,5,2048,20)
  Suggested Cell Sizes:
    HEAPP(ON,32,,80,,192,,520,,1232,,2048,)
 . . .
```

3. Take the heap values from the “Suggested Cell Sizes” line in the storage utilization report and use them in another RPTSTG(ON) function to get another report on storage utilization.
   a. In the administrative console, click **Environment** > **WebSphere variables** > **New**.
   b. Specify _CEE_RUNOPTS in the **Name** field, and RPTOPTS(ON),RPTSTG(ON),HEAPPOOLS(ON,32,,80,,192,,520,,1232,,2048,) or RPTOPTS(ON),RPTSTG(ON),HEAPP(ON,32,,80,,192,,520,,1232,,2048,) in the **Value** field.
   c. Click **Save** to save your changes

   The following example shows the servant joblog output from specifying one of these values.

   Example:

```
   .   .
0    HEAP statistics:
       Initial size:                                  83886080

       Increment size:                                 5242880
       Total heap storage used (sugg. initial size):  195803218

       Successful Get Heap requests:                    426551
       Successful Free Heap requests:                   424262
       Number of segments allocated:                         1
       Number of segments freed:                             0
   .   .   .

  Suggested Percentages for current Cell Sizes:
    HEAPP(ON,32,8,80,43,192,48,520,20,1232,5,2048,20)
  Suggested Cell Sizes:
    HEAPP(ON,32,,80,,192,,520,,1232,,2048,)
 . . .
```

4. Take the heap values from the “Suggested Percentages for current Cell Sizes” line of the second storage utilization report and use them in another RPTSTG(ON) function to get a third report on storage utilization.
   a. In the administrative console, click **Environment** > **WebSphere variables** > **New**.
   b. Specify _CEE_RUNOPTS in the **Name** field, and RPTOPTS(ON),RPTSTG(ON,32,8,80,43,192,48,520,20,1232,5,2048,20) in the **Value** field.
   c. Click **Save** to save your changes

   The following example shows the servant joblog output from specifying this value.

   Example:

```
   .   .
0    HEAP statistics:
       Initial size:                                  83886080

       Increment size:                                 5242880
       Total heap storage used (sugg. initial size):  198372130

       Successful Get Heap requests:                    426551
       Successful Free Heap requests:                   424262
       Number of segments allocated:                         1
       Number of segments freed:                             0
   .   .   .

  Suggested Percentages for current Cell Sizes:
    HEAPP(ON,32,8,80,43,192,48,520,20,1232,5,2048,20)
  Suggested Cell Sizes:
    HEAPP(ON,32,,80,,192,,520,,1232,,2048,)
 . . .
```

5. On the third storage utilization report, look for the “Total heap storage used (sugg. initial size):” line and use this value for your initial LE heap setting. For example, in the report in the third report example, this value is 198372130.
6. Remove the RPTSTG WebSphere variable from your server settings.because a small performance degradation occurs while you are collecting the storage use information.
   a. In the administrative console, click **Environment > WebSphere variables**.
   b. Select _CEE_RUNOPTS, and click **Delete**.
7. For your client programs that run on z/OS, you should specify HEAPP(ON) on the proc of your client to get the default LE heap pools. LE provides additional pools (more than 6), and larger than 2048 MB cell size in future releases of z/OS. You might be able to take advantage of these increased pools and cell sizes, if you have that service on your system.
8. If you use LE HEAPCHECK, make sure to turn it off after you verify that your code does not include any uninitialized storage. HEAPCHECK can be very expensive.

Tuning the WebSphere Application Server for z/OS runtime

To optimize performance, review the following steps involved in tuning the WebSphere Application Server for z/OS runtime.

Procedure

- “Review the WebSphere Application Server for z/OS configuration”
- “Internal tracing tips for WebSphere for z/OS” on page 58
- “Location of executable programs tips for z/OS” on page 58
- Security tuning tips for z/OS

Review the WebSphere Application Server for z/OS configuration

WebSphere Application Server provides tunable settings for its major components to enable you to make adjustments to better match the runtime environment to the characteristics of your application.

Before you begin

Many applications can run successfully without any changes to the default values for these tuning parameters. Other applications might need changes, for example, a larger heap size, to achieve optimal performance.

About this task

Before you read a description of WebSphere Application Server for z/OS tuning guidelines, it is important to note that, no matter how well the middleware is tuned, it cannot make up for poorly designed and coded

applications. Focusing on the application code can help improve performance. Often, poorly written or designed application code changes will make the most dramatic improvements to overall performance.

Procedure

1. Review the WebSphere for z/OS configuration. One simple way to do this is to look in your application control and server regions in SDSF.
2. When each server starts, the runtime prints out the current configuration data in the joblog.

Internal tracing tips for WebSphere for z/OS

WebSphere Application Server traces can be extremely helpful in detecting and diagnosing problems.

By properly setting trace options, you can capture the information needed to detect problems without significant performance overhead.

- Ensure that you are not collecting more diagnostic data than you need.

  You should check your WebSphere for z/OS tracing options to ensure that **ras_trace_defaultTracingLevel**=0 or 1, and that **ras_trace_basic** and **ras_trace_detail** are not set.

  How to view or set: Use the WebSphere administrative console:
  1. Click **Environment > Manage WebSphere Variables**.
  2. On the Configuration Tab check for any of these variables in the name field and observe the variable setting in the value field.
  3. To change or set a variable, specify the variable in the name field and specify the setting in the value field. You can also describe the setting in the description field on this tab.
- For the best performance with any level of tracing, including **ras_trace_defaultTracingLevel**=1, set **ras_trace_outputLocation** to BUFFER. This trace setting stores the trace data in memory, which is later asynchronously written to a CTRACE data set. Setting **ras_trace_outputLocation** to SYSPRINT or TRCFILE provides approximately the same level of performance, but significantly less than BUFFER.
- You can use the **ras_trace_BufferCount** and **ras_trace_BufferSize** settings to control the amount of storage used for trace buffers. Generally, the larger the buffer allocation, the better the performance. However, specifying a buffer allocation that is too large can cause a decrease in performance due to system paging.

  The default settings of **ras_trace_BufferCount**=4 and **ras_trace_BufferSize**=1M should perform sufficiently for most applications.
- Make sure you disable JRAS tracing.

  To do this, look for the following lines in the trace.dat file pointed to by the JVM properties file:

  ```
  com.ibm.ejs.*=all=disable

  com.ibm.ws390.orb=all=disable
  ```

  Ensure that both lines are set to **=disable** or delete the two lines altogether.

  Note: If **ras_trace_outputLocation** is set, you may be tracing and not know it.

Location of executable programs tips for z/OS

If you choose to not put most of the runtime in LPA, you might find that your processor storage gets a bigger workout as the load increases.

IBM recommends that you install as much of the WebSphere Application Server for z/OS code in LPA as is reasonable. Also, ensure that you have eliminated any unnecessary STEPLIBs which can affect performance. If you must use STEPLIBs, verify that any `STEPLIB DD`s in the controller and servant procs do not point to any unnecessary libraries. Refer to UNIX System Services (USS) tuning tips for z/OS for USS shared file system tuning considerations.

At a minimum, WebSphere for z/OS will start three address spaces, so that any code that is not shared will load three copies rather than one. As the load increases, many more servants may start and will contribute additional load on processor storage.

Review the `PATH` statement to ensure that only required programs are in the PATH and that the order of the PATH places frequently-referenced programs in the front.

Workload management (WLM) tuning tips for z/OS

You can use the administrative console to provide the job control language (JCL) PROC name for the servant and the JCL Parm for the servant and thereby set up a dynamic application environment. Even if you set up a dynamic application environment, you must set the WLM goals for your environment.

Proper WLM goals can significantly affect your application throughput. The WebSphere Application Server address spaces should be given a fairly high priority. When setting the WLM goals for your z/OS system, you might want to:

- Classify location service daemons and controllers as SYSSTC or high velocity.
- Use STC classification rules to classify velocity goals for application servers.Use STC classification rules to classify velocity goals for application servers.

  Early in the life of the servant, or if you set the have set **`ManageNonEnclaveWork`** parameter in IEAOPTxxWLM to `yes`:

  – Java garbage collection runs under this classification. Java garbage collection is a CPU and storage intensive process. If you set the velocity goal too high garbage collection can consume more of your system resources than desired. If your Java heap is correctly tuned, garbage collection for each servant should run no more than 5% of the time. Also, providing proper priority to garbage collection processing is necessary since other work in the servant is stopped during much of the time that garbage collection is running.

  – JavaServer Page file compiles run under this classification. If your system is configured to do these compiles at runtime, setting the velocity goal too low can cause longer delays waiting for JavaServer Page file compiles to complete.

  If the **`ManageNonEnclaveWork`** parameter in IEAOPTxxWLM is set to `no`, or is not specified, most Java garbage collection and JavaServer Page compiles are managed according to the service class to which WLM has bound the servant for the work the servant is processing.

  Application work is classified under the work manager.
- Set achievable percentage response time goals.

  For example, a goal that 80% of the work will complete in .25 seconds is a typical goal. Velocity goals for application work are not meaningful and should be avoided.
- Make your goals multi-period.

  This strategy might be useful if you have distinctly short and long running transactions in the same service class. On the other hand, it is usually better to filter this work into a different service class if you can. Being in a different service class will place the work in a different servant which allows WLM much more latitude in managing the goals.
- Define unique WLM report classes for servant regions and for applications running in your application environment. Defining unique WLM report classes enables the resource measurement facility (RMF) to report performance information with more granularity.
- Use the `wlm_maximumSRCount=`*x* and `wlm_minimumSRCount=`*y* variables to control the maximum and minimum number of servants, if the WLM configuration is set to no limit.

  To specify values for these variables, in the administrative console, click **Servers > Server Types > WebSphere application servers**, and select the appropriate application server.

  gotcha: If you specify a value for the wlm_maximumSRCount variable, the value must be greater than or equal to the number of service classes defined for this application environment. If the value is less than the number of defined service classes, timeouts might be caused because there is an insufficient number of servants available.

- Periodically review the results reported in the RMF Postprocessor workload activity report:
  - Transactions per second (not always the same as client tran rate)
  - Average response times (and distribution of response times)
  - CPU time used
  - Percent response time associated with various delays
- Watch out for work that defaults to SYSOTHER. As stated in the z/OS Information Center topic *Subsystem-Specific Performance Hints*, work in subsystems that use enclaves can suffer performance degradation if left unclassified in the service definition. If you do not add classification rules for this work in your service definition, it is automatically assigned to the SYSOTHER service class, which has a discretionary goal.

Tuning message-driven bean processing on z/OS by using WebSphere MQ as the messaging provider in ASF mode

You can tune message-driven beans processing when you are running WebSphere Application Server on the z/OS platform, where WebSphere MQ is the messaging provider, and the message-driven bean has been deployed in Application Server Facilities (ASF) mode.

Before you begin

To tune message-driven bean processing, you need to consider a variety of settings together. There are a wide range of values and possibilities to consider, because of the variety of workloads possible to run in any given server.

When a message-driven bean is mapped (that is, listening to) a queue, or to a topic through a durable subscription, a JMS message first enters into the application server in the controller, so we say the server is “listening in the controller” for these messages. The “listening in the controller” term is used throughout this description of tuning message-driven bean processing.

About this task

When you are tuning message-driven bean processing in the server, you also need to consider the tuning the entire workload for the server, and the interaction between the two.

To tune message-driven bean processing, consider all the following settings together:

- WLM service class definitions
- WebSphere Application Server workload profile selection
- Message listener service listener port settings
- JMS Connection Factory pooling settings
- WebSphere MQ Queue Manager settings

It is difficult to recommend values to select for each of these settings, given the variety of workloads that can be run in any given server. There are many possibilities to consider, including the following factors:

- The number of message-driven beans.
- The administrative configuration choices, such as whether to map two message-driven beans to the same or different listener ports.
- The importance of work for message-driven beans compared with other (HTTP, IIOP) types of work running in the server.

The following suggested settings provide a starting point, and assume that the server is configured with only one application, which consists of a single message-driven bean that is installed and running on this server.

More detailed discussions explain the rationale behind the suggestions, and describe the listener port function in more detail in the “listening in the controller” case on z/OS. Together, they can help you to make your own setting selections for your own systems and servers.

Procedure

1. Set the listener port **maximum sessions** property to at least twice the maximum number of servant worker threads available to the entire server. The value of this property determines the high threshold value (high threshold = maximum sessions), and is used by the throttle to decide when to block or allow requests.
   a. Start the administrative console.
   b. In the navigation pane, click **Servers** > **Server Types** > **WebSphere application servers->*server_name*** > **[Communications] Messaging** > **Message listener service** > **[Additional Properties] Listener Ports** > ***listener_port*** The Message listener port collection panel is displayed.
   c. Select the name of the listener port that you want to work with. The Listener port settings panel is displayed.
   d. Set the **maximum sessions** property to the value you want the message-driven bean throttle to use as its high threshold value. The suggested minimum value is computed by the formula:

      ```
      2 * (maximum number of servants) * (number of worker threads in one servant)
      ```

      Here “servants” means the same as “server instances” on the administrative console. To calculate the number of worker threads in a single servant, see the description of “Workload profile”. See the *Developing and deploying applications* PDF for more information.

   To learn more about setting the Listener Port **maximum sessions** property, see the information about message-driven beans and tuning settings on z/OS.
2. Set the WebSphere MQ queue connection factory properties.
   a. To view this administrative console page, click **Resources** > **JMS->Queue connection factories**.
   b. Select the queue connection factory specified for the listener port.
   c. From the Additional Properties, select the Connection Pool panel.
   d. Set the **Max Connections** property for the Connection Pool. Allow one connection for each message-driven bean. This property value might include message-driven beans mapped onto different listener ports, if those listener ports were each, in turn, mapped onto the same connection factory. To learn more about this setting, see the information about message-driven beans and tuning settings on z/OS.
   e. From the Additional Properties of the queue connection factory, select the Session Pool panel.
   f. Set the **Max Connections** property for the session pool. Allow one session for each worker thread in a single servant. Set this property to at least the number of worker threads available to a single servant. To learn more about this setting, see the information about message-driven beans and tuning settings on z/OS.
3. Set the WebSphere MQ-related properties. Make sure that the backing WebSphere MQ queue manager has been configured with enough resources to support the intended JMS workload coming from WebSphere Application Server (and other clients). In particular, consider your queue manager CTHREAD, IDBACK, and IDFORE parameter settings. For more information on these WebSphere MQ settings, see the WebSphere MQ information center.

Example

1. If your server is configured with the **maximum server instances** value set to `3`, (whatever the minimum number might be), and if your workload profile is `LONGWAIT` (which means that each servant contains 40 worker threads), set your listener port **maximum sessions** value to at least

   ```
   240 = 2 * 3 * 40
   ```

2. Suppose that your application contains two individual message-driven beans, each of which has an onMessage() implementation that forwards the message to another JMS destination. Therefore, each

message-driven bean needs its own JMS connection factory to complete this task. Suppose the Administrator has mapped each message-driven bean JMS connection factory resource reference onto the same administratively-defined connection factory used by the listener port that each of these message-driven beans is mapped onto.

In this case, you need to set the connection factory **Connection Pool Max Connections** value to 42. One connection for each of the two message-driven beans to be used by the listener port, and one connection potentially for each of the 40 onMessage() dispatches that night be running concurrently. (Remember that the connection pool is a per-servant pool).

3. Set the connection factory **Session Pool Max Connections** to 40, the number of worker threads in a single servant, regardless of the number of servants.

For debugging tips, refer to Optimizing MDB throttle support for debugging in z/OS.

Tuning storage

Running application servers on your z/OS system often requires a high amount of virtual storage. Because virtual storage uses real storage as backup, real storage usage might also be high. Therefore make sure that you do not underestimate the amount of virtual storage that you allocate to running your application servers.

Before you begin

Determine your application server virtual storage requirements based on the number of application servers you are running and the number of requests that each of these servers handles.

About this task

Perform one or more of the following steps if you need to improve client request throughput.

Procedure

1. Allocate additional virtual storage. The setting of REGION on the JCL for the proc controls the amount of virtual storage available to a z/OS address space. The default values for the WebSphere Application Server controller and servant are set to zero, which tells the operating system to allocate all the available region (close to 2GB). You can limit the amount of virtual storage allocated by setting the REGION parameter to a value other than zero. The size of the JVM heap is the most important factor when determining the setting of the REGION parameter. You should only need to set the REGION to something other than zero when the JVM heap size is very large. The z/OS operating system allocates user storage from the bottom of the address space, which is where the JVM heap is allocated, and system storage from the top. System abends can occur when the system tries to obtain virtual storage and none is available. A non-zero REGION parameter setting prevents this from occurring by preserving storage at the top of the address space for system use. In almost all cases running with the default REGION will be satisfactory.

   Note: For more information on REGION=0M and IEFUSI, please see Installing your application serving environment ection of the information center.
2. Convert application servers with high virtual storage usage to run in 64-bit mode. Running an application server in 64-bit mode allows you to specify larger JVM heap sizes.
3. Convert deployment mangers, that are managing cells in which very large applications are deployed, to run in 64-bit mode.
4. Allocate additional real storage. The total amount of real storage that your system requires depends on the number of servers you are running and the size of the JVM heaps for each server. You should allocate at least 512MB of real storage for a small configuration.

   Recommendation: Sometimes in a heavy use environment, 2GB of central storage is not enough to handle the real storage demands of a high volume Java application. In this situation, you might want to

configure your servers to run in 64-bit mode. Running your servers in 64-bit mode gives you the ability to dedicate more central storage to the LPAR, and the ability to define more than 2GB of central storage. When you configure your servers to run in 64-bit mode, all of the storage is defined as central storage.

The z/OS operating system running on a zSeries processor always runs in 64-bit mode. If you are using a non-zSeries processors, or are running your servers in 31-bit mode, you can minimize paging by defining more expanded storage.

Directory conventions

References in product information to *app_server_root*, *profile_root*, and other directories imply specific default directory locations. This article describes the conventions in use for WebSphere Application Server.

Default product locations - z/OS

app_server_root
Refers to the top directory for a WebSphere Application Server node.

The node may be of any type—application server, deployment manager, or unmanaged for example. Each node has its own *app_server_root*. Corresponding product variables are *was.install.root* and *WAS_HOME*.

The default varies based on node type. Common defaults are *configuration_root*/AppServer and *configuration_root*/DeploymentManager.

configuration_root
Refers to the mount point for the configuration file system (formerly, the configuration HFS) in WebSphere Application Server for z/OS.

The *configuration_root* contains the various *app_server_root* directories and certain symbolic links associated with them. Each different node type under the *configuration_root* requires its own cataloged procedures under z/OS.

The default is /wasv8config/*cell_name*/*node_name*.

plug-ins_root
Refers to the installation root directory for Web Server Plug-ins.

profile_root
Refers to the home directory for a particular instantiated WebSphere Application Server profile.

Corresponding product variables are *server.root* and *user.install.root.*

In general, this is the same as *app_server_root*/profiles/*profile_name*. On z/OS, this will always be *app_server_root*/profiles/default because only the profile name "default" is used in WebSphere Application Server for z/OS.

smpe_root
Refers to the root directory for product code installed with SMP/E or IBM Installation Manager.

The corresponding product variable is *smpe.install.root*.

The default is /usr/lpp/zWebSphere/V8R5.

Using PassByReference optimization in SCA applications

Support exists for the @AllowsPassByReference annotation, which can be used to bypass marshaling and unmarshaling when a client invokes a service located in the same JVM over a remote interface.

About this task

Typically, a performance intensive aspect of service invocations is data marshaling and unmarshaling. Though invocation over a local interface always results in pass-by-reference semantics so that no data is copied, an invocation over a remotable interface entails pass-by-value semantics, which typically results in copying of the data which can be expensive.

The SCA default binding provides the @AllowsPassByReference as an optimization that you can use on your service implementation at the class level or at the individual method level.

In placing the @AllowsPassByReference annotation on the service implementation class or methods, the implementor agrees not to modify the data in a way that would violate the pass-by-value semantics. This allows both client and service to assume they are working with their own copy of the data even though the runtime environment has optimized to not perform the actual data serialization and deserialization, to save this expense.

Parameters, return types, and business exceptions are passed by reference if the service implementation class has the @AllowsPassByReference annotation defined at the class level or individual method level.

More specifically, the PassByReference optimization is performed when all of the following are true:

- Client and service have been targeted to the same JVM.
- The invocation is over the default binding.
- @AllowsPassByReference is present. Either the service implementation is a Java implementation with an appropriate @AllowsPassByReference annotation, or a composite implementation, ultimately recursively implemented in terms of such an @AllowsPassByReference-annotated Java implementation.
- All input, output, and exception types have the same package-qualified class names and can be loaded by a class loader common to or shared by both client and service.
- Both client and service are part of the same business-level application.

  This requirement applies to OSOA SCA applications. For OSOA SCA applications, both client and service must be part of the same business-level application. This requirement does not apply to OASIS SCA applications.

Procedure

1. To enable PassByReference optimization for SCA applications, ensure all classes that you want to optimize are loaded by the same class loader. Use the SCA contribution import and export support.
2. Create a Java archive (JAR) file that contains all classes that are loaded by the same class loader during both client and service execution.
3. Add an `sca-contribution.xml` file to the `META-INF` directory in the JAR.

   See the OSOA Assembly specification for information on `sca-contribution.xml`. The contribution definition must contain an `export.java` statement that exports all packages contained in the JAR that are accessed by either the client or service JAR file. For example:

   ```
   <contribution xmlns="http://www.osoa.org/xmlns/sca/1.0"
       targetNamespace="http://test.sca.scafp/pbr/shared/java">
       <export.java package="com.ibm.sample.interface"/>
       <export.java package="com.ibm.sample.types"/>
   </contribution>
   ```

   If the client and service JAR files are not already using an `sca-contribution.xml` file, update these files to use a contribution definition that imports the packages that are exported by the shared library. For example, the contribution files for the client and service that access the previous shared contribution might look like this:

   ```
   Client:
   <contribution xmlns="http://www.osoa.org/xmlns/sca/1.0"
       targetNamespace="http://test.sca.scafp/pbr/shared"
       xmlns:pbr="http://test.sca.scafp/pbr/shared">
       <deployable composite="pbr:PassByRef.SharedClient"/>
   ```

```
    <import.java package="com.ibm.sample.interface"/>
    <import.java package="com.ibm.sample.types"/>
</contribution>

Service:
<contribution xmlns="http://www.osoa.org/xmlns/sca/1.0"
    targetNamespace="http://test.sca.scafp/pbr/shared"
    xmlns:pbrsh="http://test.sca.scafp/pbr/shared">
    <deployable composite="pbrsh:PassByRef.SharedService"/>
    <import.java package="com.ibm.sample.interface"/>
    <import.java package="com.ibm.sample.types"/>
</contribution>
```

4. Deploy the SCA application

   Add the client, service, and shared contributions as an asset into the WebSphere repository. This can occur in any order, however you must add the shared contribution as an asset before you can add either the client or service asset as a composition unit to a business-level application. Only the client and service assets need to be added as composition units to your business-level applications. During the add composition unit operation for both the client and the service, the shared asset is automatically added to the business-level application as a shared library.

   Clients that are deployed outside of a business-level application cannot use the PassByReference optimization to invoke SCA services deployed inside a business-level application. For example, a user-created web application archive (WAR) file using the default binding cannot be installed into a business-level application, and therefore a WAR-hosted client might not participate in the PassByReference optimization. The PassByReference optimization is supported only between JAR files.

   For OSOA SCA applications, you must install both service JAR files in the same business-level application.

   For OASIS SCA applications, the JAR files can be in separate business-level applications.

Results

You have enabled PassByReference optimization for SCA applications.

Tuning the application server using pre-defined tuning templates

You can use the python-based tuning script, applyPerfTuning.py, along with one of its template files, to apply pre-defined performance tuning templates to your application server or cluster. The script, and these property-based template files are located in the *WAS_HOME*/bin directory.

Before you begin

bprac: The configuration settings applied by this script and the associated tuning templates should be viewed as potential performance tuning options for you to explore or use as a starting point for additional tuning. The configuration settings that each of the pre-defined templates applies are geared towards optimizing common application server environments or scenarios. Typically, these settings improve performance for many applications.

Because optimizing for performance often involves trade-offs with features, capabilities, or functional behavior, some of these settings might impact application correctness, while other settings might be inappropriate for your environment. Please review the documentation below and consider the impact of these settings to your application inventory and infrastructure.

As with any performance tuning exercise, the settings configured by the predefined templates should be evaluated in a controlled preproduction test environment. You can then create a customized template to refine the tuning settings to meet the specific needs of your applications and production environment.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Typically, when you run the applyPerfTuning.py script, you will specify either the production.props template file or the development.props template file to apply against the target server or cluster.

- If you specify the production.props template file when you run the applyPerfTuning.py script, the script applies configuration settings that are appropriate for a production environment where application changes are rare and optimal runtime performance is important.
- If you specify the development.props template file when you run the applyPerfTuning.py script, the script applies configuration settings that are appropriate for a development environment where frequent application updates are performed and system resources are at a minimum.

In addition to these two common templates, a third template file, default.props, is provided to enable you to revert the server configuration settings back to the out-of-the-box defaults settings.

You can also create your own custom tuning template. To create a custom tuning template, copy one of the existing templates, modify the configuration settings to better fit the needs of your applications and environment, and then use the applyPerfTuning.py script to apply these customized settings. The script and properties files leverage the property file configuration management features that wsadmin provides, and can easily be augmented to tune additional server components. See the topic Using properties files to manage system configuration for more information.

About this task

Review the following table to see the configuration changes that occur based on the template file that you specify when you run the applyPerfTuning.py script. A blank cell in this table indicates that the listed parameter is not configured, or is configured back to the default settings for the server defaults.

Table 4. Tuning parameters and their template values. The table includes the tuning parameter and its value for the default template, the production template and the development template.

Parameter	Server default (default.props template file)	Production environment (production.props template file)	Development environment (development.props template file)
JVM Heap Size (MB) See the topic Tuning the IBM virtual machine for Java for more information about this setting.	50 min / 256 max	512 min / 512 max	256 min / 512 max
Verbose GC See the topic Tuning the IBM virtual machine for Java for more information about this setting.	disabled	enabled	disabled

Table 4. Tuning parameters and their template values (continued). The table includes the tuning parameter and its value for the default template, the production template and the development template.

Parameter	Server default (default.props template file)	Production environment (production.props template file)	Development environment (development.props template file)
JVM Diagnostic Trace (Generic JVM Arguments) See the topic Tuning the IBM virtual machine for Java for more information about this setting. **gotcha:** This setting might cause issues when web services are used in certain scenarios. Therefore, if you are running web services, and are experiencing throughput optimization issues, you can remove this parameter from the script, or set the opti level to `0`.	-Dcom.ibm.xml.xlxp.jaxb .opti.level=3	-Dcom.ibm.xml.xlxp.jaxb .opti.level=3	-Dcom.ibm.xml.xlxp.jaxb .opti.level=3
HTTP (9080) and HTTPS (9443) Channel maxKeepAliveRequests See the topic HTTP transport custom properties for more information about this setting.	100	10000	10000
TCP Channel maxOpenConnections	20000	500	500
TCP Channel listenBacklog	511	128	128
Development Mode See the topic Application server settings for more information about this setting.	disabled		enabled
Server Component Provisioning See the topic Application server settings for more information about this setting.	disabled	enabled	enabled
PMI Statistic Set See the topic Enabling PMI data collection for more information about this setting.	basic	none	none

Table 4. Tuning parameters and their template values (continued). The table includes the tuning parameter and its value for the default template, the production template and the development template.

Parameter	Server default (default.props template file)	Production environment (production.props template file)	Development environment (development.props template file)
Authentication Cache Timeout See the topic Authentication cache settings for more information about this setting.	10 minutes	60 minutes	60 minutes
Data Source Connection Pool Size* See the topic Connection pool settings for more information about this setting.	1 min / 10 max	10 min / 50 max	
Data Source Prepared Statement Cache Size* See the topic WebSphere Application Server data source properties for more information about this setting.	10	50	
ORB Pass-by-Reference** See the topic Request Broker service settings for more information about this setting.	disabled	enabled	enabled
Web Server Plug-in ServerIOTimeout	900	900	900
Thread Pools (Web Container, ORB, Default) See the topic Thread pool settings for more information about this setting.	50 min / 50 max, 10 min / 50 max, 20 min / 20 max		5 min / 10 max

Table notes:

* Indicates items that are tuned only if they exist in the configuration. For example, a data source connection pool typically does not exist until an application is installed on the application server. If these items are created after your run the script, they are given the standard server default values unless you specify other settings.

** Enabling ORB Pass-By-Reference can cause incorrect application behavior in some cases, because the Java EE standard assumes pass-by-value semantics. However, enabling this option can improve performance up to 50% or more if the EJB client and server are installed in the same instance, and your application is written to take advantage of these feature. The topic Object Request Broker service settings can help you determine if this setting is appropriate for your environment.

Following are a few subtle platform-specific tuning differences:

z/OS platform

The default JVM heap sizes are different than those on the other platforms:

- Default minimum heap size: 256 MB

- Default maximum heap size: 512 MB

Procedure

- Start the wsadmin tool if it is not already running, and then complete one of the following actions to tune an application server or all of the application servers in a cluster.
- Run the applyPerfTuning.py script to tune a specific server or cluster of servers running in a production environment.

  ```
  wsadmin -f applyPerfTuningTemplate.py
  [-nodeName node_name -serverName server_name][clusterName cluster_name] -templateFile production.props
  ```

- Run the applyPerfTuning.py script to tune a specific server or cluster of servers running in a development environment.

  ```
  wsadmin -f applyPerfTuningTemplate.py
  [-nodeName node_name -serverName server_name][clusterName cluster_name] -templateFile development.props
  ```

- Run the applyPerfTuning.py script to change the settings for a server or a cluster back to the standard out-of-the-box default configuration settings.

  ```
  wsadmin -f applyPerfTuningTemplate.py
  [-nodeName node_name -serverName server_name][clusterName cluster_name] -templateFile default.props
  ```

What to do next

Conduct a performance evaluation, and tuning exercise to determine if you should further fine tune the server for your specific applications.

Chapter 6. Troubleshooting performance problems

This topic illustrates that solving a performance problem is an iterative process and shows how to troubleshoot performance problems.

About this task

Solving a performance problem is frequently an iterative process of:

- Measuring system performance and collecting performance data
- Locating a bottleneck
- Eliminating a bottleneck

This process is often iterative because when one bottleneck is removed the performance is now constrained by some other part of the system. For example, replacing slow hard disks with faster ones might shift the bottleneck to the CPU of a system.

Measuring system performance and collecting performance data

- Begin by choosing a *benchmark*, a standard set of operations to run. This benchmark exercises those application functions experiencing performance problems. Complex systems frequently need a warm-up period to cache objects, optimize code paths, and so on. System performance during the warm-up period is usually much slower than after the warm-up period. The benchmark must be able to generate work that warms up the system prior to recording the measurements that are used for performance analysis. Depending on the system complexity, a warm-up period can range from a few thousand transactions to longer than 30 minutes.
- If the performance problem under investigation only occurs when a large number of clients use the system, then the benchmark must also simulate multiple users. Another key requirement is that the benchmark must be able to produce repeatable results. If the results vary more than a few percent from one run to another, consider the possibility that the initial state of the system might not be the same for each run, or the measurements are made during the warm-up period, or that the system is running additional workloads.
- Several tools facilitate benchmark development. The tools range from tools that simply invoke a URL to script-based products that can interact with dynamic data generated by the application. IBM Rational has tools that can generate complex interactions with the system under test and simulate thousands of users. Producing a useful benchmark requires effort and needs to be part of the development process. Do not wait until an application goes into production to determine how to measure performance.
- The benchmark records throughput and response time results in a form to allow graphing and other analysis techniques. The performance data that is provided by WebSphere Application Server Performance Monitoring Infrastructure (PMI) helps to monitor and tune the application server performance. See the information on why use request metrics to learn more about performance data that is provided by WebSphere Application Server. Request metrics allows a request to be timed at WebSphere Application Server component boundaries, enabling a determination of the time that is spent in each major component.

Locating a bottleneck

Consult the following scenarios and suggested solutions:

- **Scenario:** Poor performance occurs with only a single user.

  Suggested solution: Utilize request metrics to determine how much each component is contributing to the overall response time. Focus on the component accounting for the most time. Use Tivoli Performance Viewer to check for resource consumption, including frequency of garbage collections. You might need code profiling tools to isolate the problem to a specific method. See the *Administering applications and their environment* PDF for more information.
- **Scenario:** Poor performance only occurs with multiple users.

Suggested solution: Check to determine if any systems have high CPU, network or disk utilization and address those. For clustered configurations, check for uneven loading across cluster members.

- **Scenario:** None of the systems seems to have a CPU, memory, network, or disk constraint but performance problems occur with multiple users.

  Suggested solutions:
  - Check that work is reaching the system under test. Ensure that some external device does not limit the amount of work reaching the system. Tivoli Performance Viewer helps determine the number of requests in the system.
  - A thread dump might reveal a bottleneck at a synchronized method or a large number of threads waiting for a resource.
  - Make sure that enough threads are available to process the work both in IBM HTTP Server, database, and the application servers. Conversely, too many threads can increase resource contention and reduce throughput.
  - Monitor garbage collections with Tivoli Performance Viewer or the verbosegc option of your Java virtual machine. Excessive garbage collection can limit throughput.

Eliminating a bottleneck

Consider the following methods to eliminate a bottleneck:

- Reduce the demand
- Increase resources
- Improve workload distribution
- Reduce synchronization

Reducing the demand for resources can be accomplished in several ways. Caching can greatly reduce the use of system resources by returning a previously cached response, thereby avoiding the work needed to construct the original response. Caching is supported at several points in the following systems:

- IBM HTTP Server
- Command
- Enterprise bean
- Operating system

Application code profiling can lead to a reduction in the CPU demand by pointing out hot spots you can optimize. IBM Rational and other companies have tools to perform code profiling. An analysis of the application might reveal areas where some work might be reduced for some types of transactions.

Change tuning parameters to increase some resources, for example, the number of file handles, while other resources might need a hardware change, for example, more or faster CPUs, or additional application servers. Key tuning parameters are described for each major WebSphere Application Server component to facilitate solving performance problems. Also, the performance advisors page can provide advice on tuning a production system under a real or simulated load.

Workload distribution can affect performance when some resources are underutilized and others are overloaded. WebSphere Application Server workload management functions provide several ways to determine how the work is distributed. Workload distribution applies to both a single server and configurations with multiple servers and nodes.

See the *Administering applications and their environment* PDF for more information.

Some critical sections of the application and server code require synchronization to prevent multiple threads from running this code simultaneously and leading to incorrect results. Synchronization preserves correctness, but it can also reduce throughput when several threads must wait for one thread to exit the critical section. When several threads are waiting to enter a critical section, a thread dump shows these threads waiting in the same procedure. Synchronization can often be reduced by: changing the code to

only use synchronization when necessary; reducing the path length of the synchronized code; or reducing the frequency of invoking the synchronized code.

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

APACHE INFORMATION. This information may include all or portions of information which IBM obtained under the terms and conditions of the Apache License Version 2.0, January 2004. The information may also consist of voluntary contributions made by many individuals to the Apache Software Foundation. For more information on the Apache Software Foundation, please see http://www.apache.org. You may obtain a copy of the Apache License at http://www.apache.org/licenses/LICENSE-2.0.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Intellectual Property & Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
USA

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Mail Station P300
2455 South Road
Poughkeepsie, NY 12601-5400
USA
Attention: Information Requests

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Trademarks and service marks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. For a current list of IBM trademarks, visit the IBM Copyright and trademark information Web site (www.ibm.com/legal/copytrade.shtml).

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other company, product, or service names may be trademarks or service marks of others.

Index