IBM Systems - iSeries

e-business and Web serving
WebSphere Application Server - Express Version 5
Performance

Version 5 Release 4

IBM Systems - iSeries

e-business and Web serving
WebSphere Application Server - Express Version 5
Performance

Version 5 Release 4

Note

Before using this information and the product it supports, be sure to read the information in "Notices," on page 27.

Third Edition (February 2006)

This edition applies to version 5.0 of IBM WebSphere Application Server - Express for iSeries (product number 5722-IWE) and to all subsequent releases and modifications until otherwise indicated in new editions. This version does not run on all reduced instruction set computer (RISC) models nor does it run on CISC models.

Contents

Tune performance

Performance is the measure of time and resources that are required to complete a task. The goal of performance tuning is to decrease the amount of time and resources that your application server requires to process requests. Performance tuning allows your application server to complete more tasks in less time. WebSphere Application Server - Express relies on several different components to run an enterprise application. As a result, you must determine how to optimize the performance of the entire system, in addition to the performance of the individual components.

These topics provide information about performance and describe how you can tune the performance of your WebSphere Application Server - Express environment.

"Performance overview"
This topic provides information to help you understand the factors that affect performance, and links to performance management resources.

"Performance tuning guidelines" on page 5
This topic provides information about tuning performance for your enterprise applications, application server, and other components of your system.

Performance overview

Performance is the measure of time and resources that are required to complete a task. Because of the variety of resources involved, performance tuning for Web-based applications is more complex than tuning most other applications.

To understand performance management, you need to know these basic performance concepts:

- **Throughput** is the number of client requests that the application server environment can process at a given time.
- **Response time** is the elapsed time that it takes for the application server to process a client request.
- **Load** is the amount of resources, such as main storage, processor, and I/O support, that are required by the application to process all client requests at a given time.

Application server performance is determined primarily by these components:

- **Hardware resources** such as system configuration, memory pools, and subsystems
- **Web environment configuration** such as garbage collection, request queuing, and data caching
- **Application design and implementation** such as object creation, connection pooling, and data access configuration

See these topics for more information about performance and the resources that are available to help you optimize performance:

"Performance guidelines"
This topic provides general guidelines that can help you ensure optimal performance.

"Performance resources" on page 2
This topic provides links to additional references and tools that can help you optimize performance.

Performance guidelines

This page provides basic guidelines that can help you ensure optimal performance.

- Verify that you have enough system capacity. For more information, see Prerequisites for installing and running WebSphere Application Server - Express V5.0 for iSeries in the *Installation* topic. You can also use the IBM eServer Workload Estimator or obtain professional services.
- Verify that hardware resources are allocated efficiently.
  - Provide sufficient memory and activity level in the *BASE memory pool.
  - If availability and performance are high priorities, it is recommended that you allocate WebSphere Application Server - Express processing to a separate storage pool.

  You can adjust memory pools manually or automatically.
- Verify that you have the latest WebSphere Application Server - Express Group PTF. For information about PTFs and to download the most recent Group PTF, see WebSphere Application Server for iSeries: PTFs.
- Tune the JVM garbage collector. For information about garbage collection, see Performance resources (page 3).
- Configure queues for your application server components. See these topics for more information:
  - "Queuing network" on page 10
  - Thread pool settings
  - Connection pool settings
- If your application server uses public key encryption, use hardware accelerators to improve performance. For information about the hardware accelerators that are available for iSeries, see the appropriate document based on your version of i5/OS:
  - For V5R2: iSeries Cryptographic Offload Performance Considerations
- Design applications to optimize performance. Performance resources (page 3) includes links to resources that can help you develop efficient Java applications.
- Use efficient SQL and JDBC practices to optimize data access.

Performance resources

These resources additional information and about performance:

- WebSphere Application Server performance resources (page 2)
- Java(TM) performance resources (page 3)
- iSeries performance resources (page 3)
- Performance tools (page 4)

WebSphere Application Server performance resources

WebSphere Application Server 5.0 for iSeries Performance Considerations

This page provides links to information about basic performance considerations for WebSphere Application Server. Although this information is intended for WebSphere Application Server Version 5.0, you can apply many of the same general practices to optimize performance for WebSphere Application Server - Express Version 5.0.

WebSphere and Java tuning tips

This page provides links to papers and articles that can help you take advantage of the latest iSeries performance improvements, tools, and tuning methods to optimize WebSphere Application Server - Express performance.

Java(TM) performance resources

Tuning Garbage Collection for Java(TM) and WebSphere on iSeries

This PDF manual describes how to configure garbage collection for Java applications that run in WebSphere Application Server - Express. You can find additional information about garbage collection at IBM developerWorks.

Performance Documentation for the Java Platform

Sun Microsystems, Inc. provides links to performance documentation for the Java Platform.

Optimize your Java application's performance

This article examines the optimization process as a whole, rather than focusing on a single technique. You can also use the search function of the IBM developerWorks Web site to find additional information about Java application performance.

Basic Java Performance for iSeries

This white paper explains JIT-MMI, the user classloader verification cache, and memory pool considerations.

iSeries performance resources

Tune Java program performance with the IBM Developer Kit for Java
This Information Center topic describes how you can use the IBM Developer Kit for Java to optimize the performance of your Java applications.

Performance management

This Web site provides extensive information about iSeries performance management.

Performance Management Resource Library

This page provides links to information that you can use to optimize server performance. The resource library includes white papers, articles, tools documentation, and more.

Performance Capabilities Reference Manual

The Performance Management Library provides links to several editions of the Performance Capabilities Reference Manual. This manual includes information about optimizing performance for DB2 UDB for iSeries, Web servers and WebSphere products, and Java applications.

Performance
This Information Center topic provides extensive information about managing and tuning the performance of your iSeries server.

Performance tools

Java and WebSphere Performance on IBM eServer iSeries Servers

This Redbook provides tips, techniques, and methodologies for working with Java and WebSphere Application Server - Express performance. The specific performance measurements this Redbook are based on versions 3.5 and 4.0 of WebSphere Application Server and i5/OS V5R1. However, the document might provide useful information about general performance concepts and techniques.

WebSphere Application Server Performance Tuning and Analysis Tools

This high-level online course is offered through IBM eServer Solutions Enablement. The course describes elements that can affect WebSphere Application Server - Express performance, provides basic guidelines to help you achieve optimal performance, and discusses various available tools that you can use to analyze and tune performance.

Heap Analysis Tools for Java^(TM)

This tool is a component of the iDoctor for iSeries suite of performance monitoring tools. The Heap Analysis Tools component performs Java application heap analysis and object create profiling (size and identification) over time. This tool is sometimes called Java Watcher or Heap Analyzer.

Applications for performance management
This Information Center topic provides links to several tools that you can use to monitor and manage iSeries performance.

The ANZJVM (Analyze Java Virtual Machine) command
The ANZJVM command collects information about the Java Virtual Machine (JVM) for a specified job. This command is available in i5/OS V5R2 and later.

The Dump Java Virtual Machine (DMPJVM) command
This command dumps JVM information for a specified job.

Performance Tools for iSeries
Performance Tools for iSeries is a set of tools and commands that you can use to view and analyze iSeries performance data in several ways. See this Information Center topic for more information.

Performance Explorer (PEX) Information Center topic
Performance explorer is a data collection tool that helps you identify the causes of performance problems that cannot be identified with the other available tools or general trend analysis. See this Information Center topic for more information.

Performance Explorer (PEX) Web site

This Web site provides additional information about PEX.

Performance Trace Data Visualizer

Performance Trace Data Visualizer (PTDV) for iSeries is a tool for processing, analyzing, and viewing Performance Explorer collection data residing in PEX database files.

Performance Data Collector tool
The Performance Data Collector (PDC) tool provides profile information about the programs that run on the iSeries server. See this Information Center topic for more information.

Collection Services
You can use Collection Services to collect performance data, which you can analyze with other performance tools. See this Information Center topic for more information.

IBM Performance Management for eServer iSeries Information Center topic
IBM Performance Management for iSeries (formerly known as PM/400) uses Collection Services to gather the nonproprietary performance and capacity data from your server and then sends the data to IBM for analysis. See this Information Center topic for more information.

IBM Performance Management for eServer iSeries Web site

This Web site provides additional information about PM for eServer iSeries.

iDoctor for iSeries

iDoctor for iSeries is a suite of applications that can help you monitor performance and troubleshoot common problems on your iSeries server.

Performance tuning guidelines

Each application server instance has several parameters that can influence application performance. You can use the administrative console to configure and tune applications, Web containers, and application servers in the administrative domain. You can also configure settings for other components of your iSeries environment to optimize performance.

- WebSphere Application Server - Express tuning parameters (page 5)
- Additional tuning parameters (page 6)

WebSphere Application Server - Express tuning parameters

These topics describe tuning parameters that are specific to WebSphere Application Server - Express and application server instances.

"Application server tuning parameters"
You can tune parameters for several application server components, such as the Web container and the Object Request Broker. This topic provides information about tuning these parameters.

"Queuing network" on page 10
This topic describes how to tune the components of the queueing network to optimize performance.

"Web services tuning tips" on page 13
This topic describes considerations for tuning Web services.

"Performance tips for wsadmin" on page 14
This topic provides tips to help you run wsadmin commands more efficiently.

Enabling security decreases performance. You can tune your security configuration to minimize this impact. For more information, see Tune your security configuration in *Security*.

Additional tuning parameters

These topics provide information about other components of your environment that can affect WebSphere Application Server - Express performance:

"Hardware capacity and configuration" on page 15
This topic provides information about configuring your hardware for optimal performance.

"Java virtual machine tuning parameters" on page 16
This topic describes tuning parameters for the i5/OS Java virtual machine and other tips for optimizing Java application performance.

"Web server tuning parameters" on page 23
This topic provides information about optimizing the performance of your Web server.

"Database tuning parameters" on page 24
This topic provides information about tuning database performance.

"TCP/IP buffer sizes" on page 25
This topic describes how TCP/IP buffer sizes can affect performance.

You an also configure some settings during application assembly to optimize application performance. See "Application assembly performance checklist" on page 26 for more information.

Application server tuning parameters

You can tune application server settings to control how an application server provides services for running applications and their components. Each application server instance contains interrelated components, called a queuing network, that must be properly tuned to support the specific needs of your e-business application. The queuing network helps the system achieve maximum throughput and maintain the overall stability of the system. For more information about the queuing network, see "Queuing network" on page 10.

You can tune the following application server settings:
- Web container (page 7)
- Session management (page 8)
- Data sources (page 8)

- Process Priority (page 9)

Web container

Each application server includes a Web container. The application server routes servlet requests along a transport queue between the Web server plug-in and the Web container. Default Web container properties are set for simple Web applications. However, these values might not be appropriate for more complex Web applications. You can adjust these parameters to tune the Web container based on the specific needs of your environment:

- **Thread pool Maximum size**
  - Description: This value limits the number of requests that your application server can process concurrently. For more information, see Thread pool settings.

    Note: The information on the settings page applies to all thread pools in WebSphere Application Server - Express.
  - How to view or set:
    1. Start the administrative console.
    2. In the topology tree, expand **Servers** and click **Application Servers**.
    3. Click the name of the application server that you want to configure.
    4. Click **Web Container**.
    5. On the **Web Container** page, click **Thread Pool**.
    6. Specify a value for the **Maximum size** field.
    7. Click **Apply** or **OK**.
    8. Save the configuration.
    9. Stop and restart the application server.
  - Default value: 50
  - Recommended value: This value should be set to handle the peak load on your application server. It is recommended that you specify a maximum size less than or equal to the number of threads processing requests in your HTTP server. A value in the range 25-50 is generally a good starting point.
- **Growable thread pool**
  - Description: This setting specifies whether the number of threads can increase beyond the maximum size configured for the thread pool. For more information, see Thread pool settings.
  - How to view or set:
    1. Start the administrative console.
    2. In the topology tree, expand **Servers** and click **Application Servers**.
    3. Click the name of the application server that you want to configure.
    4. Click **Web Container**.
    5. On the **Web Container** page, click **Thread Pool**.
    6. Select **Allow thread allocation beyond maximum thread size**.
    7. Click **Apply** or **OK**.
    8. Save the configuration.
    9. Stop and restart the application server.
  - Default value: Disabled

- Recommended value: It is recommended that you do not enable this property if you are confident the thread pool maximum size is large enough to adequately process the peak load on your application server. If you want to allow the thread pool to exceed the configured maximum pool size, enable this property. This setting is beneficial if the application server receives an unexpected increase in requests or if the maximum pool size is set too low. In this scenario, additional threads are created to handle the increased number of requests. These connections are destroyed when the number of requests returns to its typical level. However, enabling the growable thread pool setting might cause a large number of threads to be created, and can have a negative impact on system storage and performance.

You can also tune several custom parameters for HTTP transports in the Web container. For more information, see Set custom properties for an HTTP transport.

Session management

The installed default settings for session management are configured for optimal performance. For more information, see Tune session management in *Application development* and Tuning parameter settings.

Data sources

Applications uses data sources to access databases. The following data source settings can affect performance:

- **Connection pooling** For moreinformation about connection pooling, see Connection pooling in *Application Development* and Connection pool settings.

  - Maximum connection pool
    - Description: This value specifies the maximum number of managed connections for a pool.
    - How to view or set:
      1. Start the administrative console.
      2. In the topology tree, expand **Resources** and click **JDBC Providers**.
      3. Click the name of the provider for the data source that you want to configure.
      4. Click **Data Sources**.
      5. Click the name of the data source that you want to configure.
      6. Click **Connection Pool**.
      7. Specify a value in the **Max Connections** field.
      8. Click **Apply** or **OK**.
      9. Save the configuration.
      10. Stop and restart the application server.
    - Default value: 10
    - Recommended value: Set the value for the connection pool lower than the value for the **Max Connections** option in the Web container (page 7). If the pool is larger than necessary, it might waste memory and other system resources. A setting of 10-25 is suitable for many applications. For additional about connection pool size, see “Queuing network” on page 10.
  - Minimum connection pool
    - Description: This value specifies the minimum number of managed connections for a pool.
    - How to view or set:
      1. Start the administrative console.
      2. In the topology tree, expand **Resources** and click **JDBC Providers**.

        3. Click the name of the provider for the data source that you want to configure.
        4. Click **Data Sources**.
        5. Click the name of the data source that you want to configure.
        6. Click **Connection Pool**.
        7. Specify a value in the **Min Connections** field.
        8. Click **Apply** or **OK**.
        9. Save the configuration.
        10. Stop and restart the application server.
    - Default value: 1
    - Recommended value: Set the minimum pool size to handle the average load on the system.
- **Statement cache size**
  - Description: WebSphere Application Server - Express provides a statement cache for each data source. This value represents the number of free prepared statements per connection in the connection pool. The statement cache stores query information for the data source. You can adjust to size of the statement cache to optimize performance. For more information, see Data source settings.
  - How to view or set:
    1. Start the administrative console.
    2. In the topology tree, expand **Resources** and click **JDBC Providers**.
    3. Click the name of the provider for the data source that you want to configure.
    4. Click **Data Sources**.
    5. Click the name of the data source that you want to configure.
    6. Specify a value in the **Statement Cache Size** field.
    7. Click **Apply** or **OK**.
    8. Save the configuration.
    9. Stop and restart the application server.
  - Default value: 10
  - Recommended value: In most situations, it is recommended that you set this to the number of unique statements for each application that uses the datasource. Setting the cache size to this value avoids cache discards, and generally results in the best performance. However, if the cache is too large, it might cause performance problems as a result of increased cache management and increased use of system resources. If you have a large number of unique statements, a smaller number might be appropriate. For information about tuning the statement cache size, see Tuning the WebSphere Prepared Statement Cache.

Process Priority

- Description: The priority setting establishes the application server job's run priority. The default process priority is 25. The application server does not override the default behavior of Java thread creation. Worker threads within the server are configured to run at 6 levels lower than the job's run priority. Therefore, by default, the priority of the worker threads is 31. For more information about this setting, see Process execution settings.
- How to view or set:
  1. Start the administrative console.

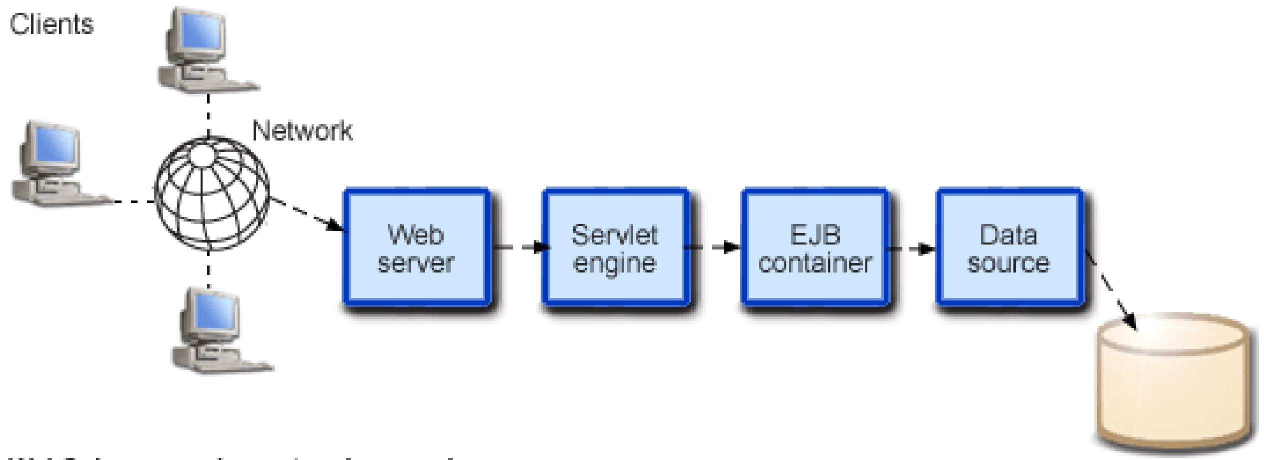
2. In the topology tree, expand **Servers** and click **Application servers**.
3. Click the name of the application server that you want to configure.
4. Click **Process Definition**.
5. Click **Process Execution**.
6. On the **Process Execution** page, specify a value in the **Process Priority** field.
7. Click **Apply** or **OK**.
8. Save the configuration.
9. Stop and restart the application server.

- Default value: 25
- Recommended value: In most situations, the default value is acceptable. However, if other workloads are running at a higher priority (that is, with a lower priority number), you might need to adjust the application server's priority so that it can more easily access the necessary resources.

Queuing network

WebSphere Application Server - Express contains interrelated components that must be tuned to support the custom needs of your e-business application. These adjustments help the system achieve maximum throughput while maintaining the overall stability of the system. This group of interconnected components is known as a queuing network. These queues or components include the network, Web server, Web container, data source, and possibly a connection manager to a custom back-end system. Each of these resources represents a queue of requests waiting to use that resource. Various queue settings include:

- "Web server tuning parameters" on page 23: ThreadsPerChild
- Web container (page 7): Thread pool maximum size, HTTP transports MaxKeepAliveConnections, and MaxKeepAliveRequests
- Data source (page 8): Connection pooling and Statement cache size



WebSphere queuing network example

Most of the queues that make up the queuing network are closed queues. A closed queue places a limit on the maximum number of requests present in the queue, while an open queue has no limit. A closed queue supports strict management of system resources. For example, the Web container thread pool setting controls the size of the Web container queue. If the average servlet running in a Web container creates 10MB of objects during each request, a value of 100 for thread pools limits the memory consumed by the Web container to 1GB.

In a closed queue, requests can be active or waiting. An active request is doing work or waiting for a response from a downstream queue. For example, an active request in the Web server is doing work, such as retrieving static HTML, or waiting for a request to be processed in the Web container. A waiting request is waiting to become active. The request remains in the waiting state until one of the active requests leaves the queue.

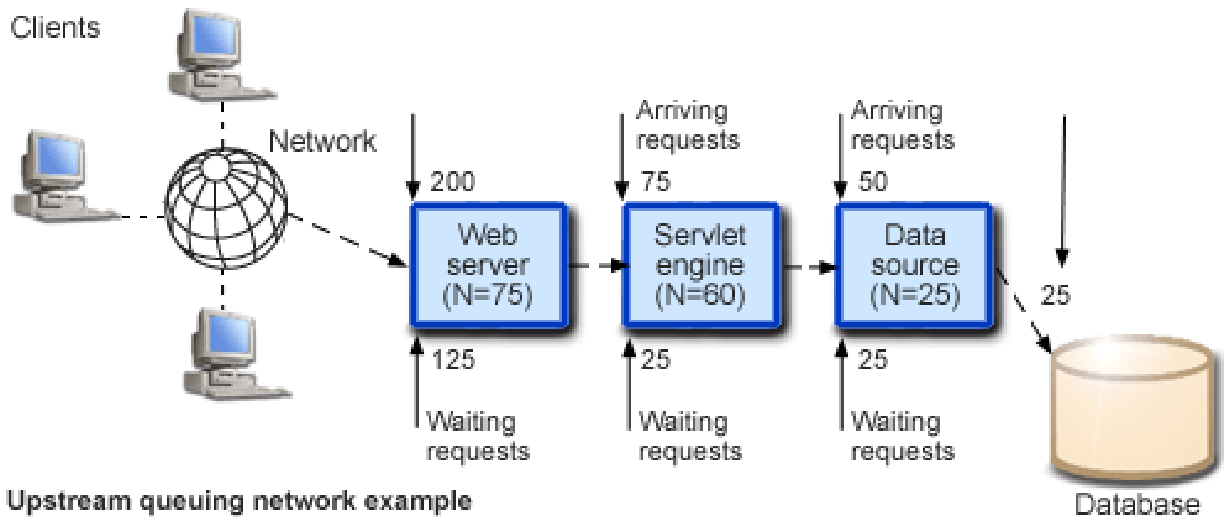
All of the Web servers that WebSphere Application Server - Express supports are closed queues, as are WebSphere Application Server - Express data sources. You can configure Web containers as open or closed queues. In general, it is recommended that you use closed queues. If there are no threads available in the pool, a new thread is created for the duration of the request.

"Queue configuration tips"
This topic provides tips to help you tune the queuing network configuration.

Queue configuration tips

This page outlines a methodology for configuring the WebSphere Application Server - Express queues. Moving the database server onto another machine or providing more powerful resources, such as a faster set of CPUs with more memory, can dramatically change the dynamics of your application server environment.

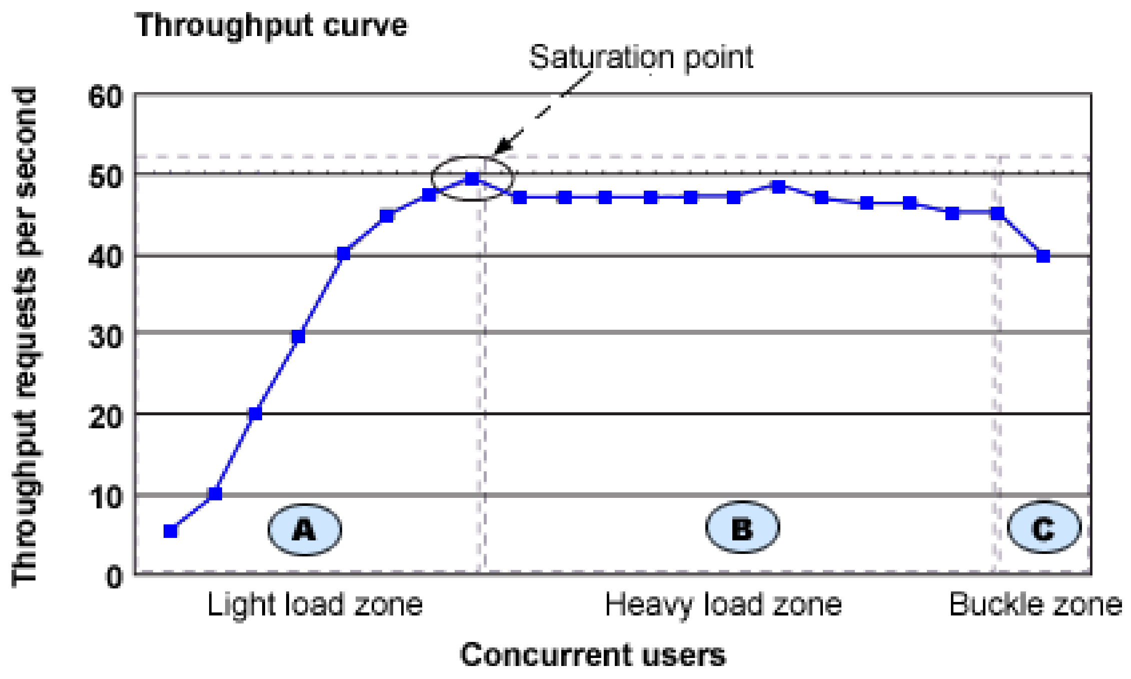
- **Minimize the number of requests in WebSphere Application Server - Express queues.**
  Performance is usually improved if requests wait in the network, ahead of the Web server, rather than waiting in the application server. That is, only requests that can be processed enter the queuing network. To achieve this result, set the size of upstream (closest to the client) queues large, and specify progressively smaller sizes for downstream (further from the client) queues. This figure provides an example of this configuration:



  Queues in the queuing network become progressively smaller as work flows downstream. In this example, 200 client requests arrive at the Web server. 125 requests remain queued in the network because the Web server is set to handle 75 concurrent requests. As the 75 requests pass from the Web server to the Web container, 25 requests remain queued in the Web server and the remaining 50 are handled by the Web container. This process progresses through the data source until 25 user requests arrive at the final destination, the database server. Because there is work waiting to enter a component at each point upstream, no component in this system must wait for work to arrive. Most of the requests wait in the network, outside of WebSphere Application Server - Express. This type of configuration adds stability, because no component is overloaded.

- **Draw throughput curves to determine when the system capabilities are maximized.**
  To run a test case that represents the full use of the production application, exercise all meaningful code paths or use the production application. Run a set of tests to determine when the system capabilities are fully stressed or when the network reaches the saturation point. Conduct these tests after most bottlenecks are removed from the application. The goal of these tests is to drive CPUs to near 100% utilization. For maximum concurrency through the system, start the initial baseline

experiment with large queues. For example, start the first experiment with a queue size of 100 at each of the servers in the queuing network: Web server, Web container, and data source. Begin a series of experiments to plot a throughput curve, increasing the concurrent user load after each experiment. For example, perform experiments with 1, 2, 5, 10, 25, 50, 100, 150 and 200 users. After each test, record the throughput requests per second, and response times in seconds per request. The curve resulting from the baseline experiments resembles the following typical throughput curve:



WebSphere Application Server - Express throughput is a function of the number of concurrent requests present in the total system. Section A, the light load zone, shows that as the number of concurrent user requests increases, the throughput increases almost linearly with the number of requests. Under light loads, concurrent requests face very little congestion within the WebSphere Application Server - Express system queues. At some point, congestion starts to develop and throughput increases at a much lower rate until it reaches a saturation point that represents the maximum throughput value, as determined by some bottleneck in the WebSphere Application Server - Express system. The most manageable type of bottleneck occurs when the WebSphere Application Server - Express machine CPUs become fully utilized. To resolve this bottleneck, add processing power.

In the heavy load zone, Section B, as the concurrent client load increases, throughput remains relatively constant. However, the response time increases proportionally to the user load. That is, if the user load is doubled in the heavy load zone, the response time doubles. At some point, represented by Section C, the buckle zone, one of the system components becomes exhausted. At this point, throughput starts to decrease. For example, the system might enter the buckle zone when the network connections at the Web server exhaust the limits of the network adapter or if the requests exceed operating system limits for file handles.

If the saturation point is reached by driving CPU utilization close to 100%, you can move on to the next step. If the saturation point occurs before system utilization reaches 100%, another bottleneck is probably the cause. For example, the application might be creating Java objects and causing excessive garbage collection bottlenecks in the Java code.

There are two ways to manage application bottlenecks: remove the bottleneck or clone the bottleneck. The best way to manage a bottleneck is to remove it. You can use a Java-based application profiler to examine overall object utilization. For a list of available tools, see Performance tools (page 4).

- **Decrease queue sizes as requests move downstream from the client.**
  The number of concurrent users at the throughput saturation point represents the maximum concurrency of the application. For example, if the application saturates WebSphere Application Server

- Express at 50 users, using 48 users might produce the best combination of throughput and response time. This value is called the Max Application Concurrency value. Max Application Concurrency becomes the preferred value for adjusting the WebSphere Application Server - Express system queues. Remember, it is desirable for most users to wait in the network; therefore, queue sizes should decrease when moving downstream farther from the client. For example, given a Max Application Concurrency value of 48, you might start with system queues at the following values: Web server 75, Web container 50, data source 45. Perform a set of additional tests with slightly higher and lower values to find the best settings.

- **Adjust queue settings to correspond to access patterns.**
  In many cases, only a fraction of the requests that pass through one queue enter the next queue downstream. For example, on a Web site with many static pages, a number of requests are fulfilled at the Web server and are not passed to the Web container. In this case, the Web server queue can be significantly larger than the Web container queue. In the previous example, the Web server queue was set to 75, rather than closer to the value of Max Application Concurrency. You can make similar adjustments when different components have different execution times.

  For example, in an application that spends 90% of its time in a complex servlet and only 10% of its time making a short Java database connectivity (JDBC) query, on average 10% of the servlets are using database connections at any time, so the database connection queue can be significantly smaller than the Web container queue. Conversely, if the majority of servlet execution time is spent making a complex query to a database, consider increasing the queue values at both the Web container and the data source. Always monitor the CPU and memory utilization for both the WebSphere Application Server - Express and the database servers to verify that the CPU or memory are not overloaded.

Web services tuning tips

Web services performance is affected primarily by these characteristics of the XML documents that are sent to Web services:

- **Size** refers to the length of data elements in the XML document.
- **Complexity** refers to number of elements that the XML document contains.
- **Level of nesting** refers to objects or collections of objects that are defined within other objects in the XML document, as in this example:

```
<primaryObject>
  <groupObject>
    <singleObject/>
    <singleObject/>
  </groupObject>
</primaryObject>
```

In addition, a Web services engine contains three major pressure points that define the performance of Web services:

- **Parsing** (Input): When a request is received, the Web services engine parses the input. There are two major performance components in parsing:
  – scanner
  – symbol or name identification
- **XML-to-object deserialization** (Input): As the document is parsed the XML input is deserialized and converted into business objects that are presented as business object parameters to the Web services. The Web services provider, which is a JavaBean provider, is not aware of its participation in a Web service.
- **Object-to-XML serialization** (Output): After the request is processed, reply is serialized into an XML document. Large documents or complex objects can affect output serialization.

Web services best practices

To optimize Web services performance, follow these general guidelines:

- **Use WebSphere Application Server - Express Web services instead of SOAP**
  The WebSphere Application Server - Express Web services implementation performs better than the SOAP implementation based on Apache SOAP. WebSphere Application Server - Express includes support for the Apache SOAP implementation so that you can run existing Web services applications.
- **Avoid large or complex XML documents**
  The performance of Web services is directly related to the size and complexity of the XML document that is transferred. As input documents increase in size or number of elements, they require more processing for parsing and deserialization. As output documents increase in size or number of elements, they require more processing for serialization.
- **Avoid small, frequent requests**
  By definition, every Web services request is a remote request. These requests usually involve the Web container in addition to the XML overhead of parsing and deserialization. If you need to send or retrieve a 50K object that has 10 properties that are each 5K long, you can retrieve the object in several ways, such as:
  - As one 50K request
  - As 10 5K requests
  - As 50 1K requests

  Because of the overhead associated with the Web container, it is more efficient to transfer a single 50K request than several smaller requests.
- **Limit the level of nesting in XML documents**
  Increasing the level of object nesting results in an increase in the number of objects that are deserialized and created when a request is processed. An object that is composed only of primitive types or strings is processed more efficiently than an object of a similar size that is composed of deeply nested Java objects.
- **Use WebSphere Application Server - Express custom serializers**
  The WebSphere Application Server - Express Web services engine provides serialization and deserialization helpers that improve runtime performance for business objects. These custom serializer and deserializer helpers specifically describe an object's properties. As a result, the Web services runtime consumes fewer resources to obtain information about the object.
- **When possible, use literal encoding instead of SOAP encoding**
  Literal and SOAP encoding are alternate forms for encoding Web services requests and responses. Each element in the SOAP body includes the XML schema definition type of the data element. SOAP encoding requires this information to make the XML document self-defining. SOAP encoding with the embedded data types increases the amount of data transferred in the Web services request.
- **Use descriptive but short property names**
  Web services is an XML text-based exchange protocol, and the names of variables and properties are included in the XML for the SOAP body portion of the message. Longer property names increase the size of the XML document.

Performance tips for wsadmin

- When you run a command from wsadmin, a new process is created with a new Java virtual machine (JVM). If you run multiple wsadmin -c commands from a batch file or a shell script, each command must run in its own JVM. The -f option creates only one process and JVM, and the Java classes are loaded only once, regardless of how many commands are in the file.

  This example invokes multiple application installation commands:

  ```
  wsadmin -c "$AdminApp install /home/myApps/App1.ear {-appname appl1}"
  wsadmin -c "$AdminApp install /home/myApps/App2.ear {-appname appl2}"
  wsadmin -c "$AdminApp install /home/myApps/App3.ear {-appname appl3}"
  ```

  To improve performance, you can run the wsadmin command with the -f option and specify a file that contains the installation commands. For example, you can create a file called appinst.jacl that contains these commands:

```
$AdminApp install /home/myApps/App1.ear {-appname appl1}
$AdminApp install /home/myApps/App2.ear {-appname appl2}
$AdminApp install /home/myApps/App3.ear {-appname appl3}
```

To invoke the file, run this command:

```
wsadmin -f appinst.jacl
```

- The AdminControl queryNames and completeObjectName commands can consume a large amount of resources in more complex topologies. For example, if you run a single instance that contains only a few MBeans, the `$AdminControl queryNames *` command performs well. However, if a scripting client connects to the deployment manager in a multiple machine environment, use a command only if it is necessary for the script to obtain a list of all of the MBeans in the system. If you need the MBeans on a specific node, it is recommended that you run the command as follows:

  ```
  $AdminControl queryNames node=myNode,*
  ```

  In this example, the command returns a list of only the MBeans on myNode.

Hardware capacity and configuration

These parameters include considerations for selecting and configuring the hardware on which the application servers can run:

- **Disk speed**
  Disk speed and the number of disk arms can have a significant effect on application server performance in these cases:
  - Your application is heavily dependent on database support
  - Your application uses messaging extensively

  In these situations, it is recommended that you use disk I/O subsystems that are optimized for performance, such as a RAID array. Distribute the disk processing across as many disks as possible to avoid contention issues that typically occur with 1 or 2 disk systems. For more information about disk arms and how they can affect performance, see iSeries Disk Arm Requirements.

- **Processor speed**
  In the absence of other bottlenecks, increasing the processing power can improve throughput, response times, or both. On iSeries, processing power can be related to the Commercial Processing Workload (CPW) value of the system. For more information about CPW values, see the Performance Capabilities Reference Manual in the Performance Management Resource Library.

- **System memory**
  If a large number of page faults occur, performing these tasks might improve performance:
  - Increase the memory available to WebSphere Application Server - Express.
  - Move WebSphere Application Server - Express to another memory pool.
  - Remove jobs from the WebSphere Application Server - Express memory pool.

  To determine the current page fault level, run the Work with System Status (WRKSYSSTS) command from an CL command line. For information about the minimum memory requirements, see iSeries and AS/400 hardware requirements in the *Installation* topic.
- **Storage pool activity levels**
  Verify that the activity levels for storage pools are sufficient. Increasing these values can prevent threads from transitioning into the ineligible condition.
  - To modify the activity level for the storage pool in which you are running WebSphere Application Server - Express, run the Work with System Status (WRKSYSSTS) command from an CL command line:

    ```
    WRKSYSSTS ASTLVL(*INTERMED)
    ```

– Set the system value QMAXACTLVL to a value equal to or greater than the total activity level for all pools, or *NOMAX.
  1. Run the Work with System Value (WRKSYSVAL) command from an i5/OS comand line:
     ```
     WRKSYSVAL SYSVAL(QMAXACTLVL)
     ```
  2. Adjust the value in the **Max Active** column.

- **Networks**
  Run network cards and network switches at full duplex. Running at half duplex decreases performance. Verify that the network speed can accommodate the required throughput. On 10/100 Ethernet networks, verify that 100MB is in use. You might also want to monitor the IOP utilization. For information about IOP utilization, see the Performance Capabilities Reference in the Performance Management Resource Library.

Java virtual machine tuning parameters

Because the application server is a Java process, it requires a Java virtual machine (JVM) to run, and to support the Java applications running on it. For more information about JVM settings, see Java virtual machine settings.

For additional tuning information, see "Java memory tuning tips" on page 18. If your application experiences slow response times at startup or first touch, you may want to consider using the Java user classloader cache. For more information, see Java cache for user classloaders in the *Programming* topic.

You can adjust these settings that determine how the system uses the JVM:

- **Class garbage collection (-Xnoclassgc)**
  – Description: This argument disables class garbage collection so that your applications can reuse classes more easily. You can monitor garbage collection using the -verbosegc configuration setting because its output includes class garbage collection statistics.
  – How to view or set:
    1. Start the administrative console.
    2. In the topology tree, expand **Servers** and click **Application Servers**.
    3. Click the name of the application server that you want to configure.
    4. Click **Process Definition**.
    5. On the **Process Definition** page, click **Java Virtual Machine**.
    6. In the **Generic JVM arguments** field, type `-Xnoclassgc`.
    7. Click **Apply** or **OK**.
    8. Save the configuration.
    9. Stop and restart the application server.
  – Default value: By default, class garbage collection is enabled.
  – Recommended value: Do not disable class garbage collection.
- **Initial heap size**
  – Description: The initial heap size specifies how often garbage collection runs, and can have a significant effect on performance. For more information, see Tuning Garbage Collection for Java(TM) and WebSphere on iSeries.
  – How to view or set:

    1. Start the administrative console.
    2. In the topology tree, expand **Servers** and click **Application Servers**.
    3. Click the name of the application server that you want to configure.
    4. Click **Process Definition**.
    5. On the **Process Definition** page, click **Java Virtual Machine**.
    6. In the **Initial Heap Size** field, specify a value.
    7. Click **Apply** or **OK**.
    8. Save the configuration.
    9. Stop and restart the application server.
  – Default value: For iSeries, the default value is 96.
  – Recommended value: 96MB per processor
- **Maximum heap size**
  – Description: This parameter specifies the maximum heap size available to the JVM code, in megabytes.
  – How to view or set:
    1. Start the administrative console.
    2. In the topology tree, expand **Servers** and click **Application Servers**.
    3. Click the name of the application server that you want to configure.
    4. Click **Process Definition**.
    5. On the **Process Definition** page, click **Java Virtual Machine**.
    6. The value is displayed in the **Maximum Heap Size** field.
    7. Click **Apply** or **OK**.
    8. Save the configuration.
    9. Stop and restart the application server.
  – Default value: For iSeries, the default value is 0. A value of 0 specifies that there is no maximum value.
  – Recommended value: It is recommended that you do not change the maximum heap size. When the maximum heap size triggers a garbage collection cycle, the iSeries JVM's garbage collection stops operating asynchronously. As a result, the application server cannot process user threads until the garbage collection cycle ends, and performance is significantly lower.
- **Just-In-Time (JIT) compiler**
  – Description: A Just-In-Time (JIT) compiler is a platform-specific compiler that generates machine instructions for each method as needed. For more information, see Using the Just-In-Time compiler and Just-In-Time compiler in the *IBM Developer Kit for Java* topic.
  – How to view or set:
    1. Start the administrative console.
    2. In the topology tree, expand **Servers** and click **Application Servers**.
    3. Click the name of the application server that you want to configure.
    4. Click **Process Definition**.
    5. On the **Process Definition** page, click **Java Virtual Machine**.
    6. If you want to disable JIT, select the checkbox for **Disable JIT**.
    7. Click **Apply** or **OK**.
    8. Save the configuration.
    9. Stop and restart the application server.
  – Default value: By default, JIT is enabled.

– Recommended value: It is recommended that you do not disable the JIT compiler. The os400.jit.mmi.threshold can have a significant effect on performance. For more information about the JIT compiler and the os400.jit.mmi.threshold property, see Just-In-Time compiler in the *IBM Developer Kit for Java* topic.

Java memory tuning tips

Enterprise applications written in the Java language involve complex object relationships and utilize 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. For more information, see WebSphere and Java tuning tips.

- Verify that the application is not creating a large number of short-lived objects.
- Verify that the application is not leaking objects.
- Verify that the Java heap parameters are set properly to handle a given object usage pattern.

Understanding the effect of garbage collection is necessary to apply these management techniques.

See these sections for more information:
- The Java garbage collection bottleneck (page 18)
- The garbage collection gauge (page 18)
- Detecting large numbers of short-lived objects (page 19)
- Detecting memory leaks (page 20)
- Java heap parameters (page 21)

The Java garbage collection bottleneck

Examining Java garbage collection can help you understand how the application is utilizing memory. Because Java provides garbage collection, your application does not need to manage server memory. As a result, 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 over-utilizing objects.

Garbage collection normally consumes from 5% to 20% of total execution time of a properly functioning application. If you do not manage garbage collection, it can have a significant negative impact on application performance, especially when running on symmetric multiprocessing (SMP) server machines.

The i5/OS JVM uses concurrent (asynchronous) garbage collection. This type of garbage collection results in shorter pause times and allows application threads to continue processing requests during the garbage collection cycle.

Garbage collection in the i5/OS JVM is controlled by the heap size settings. The initial heap size is a threshold that triggers new garbage collection cycles. If the initial heap size is 10 MB, for example, then a new collection cycle is triggered as soon as the JVM detects that 10 MB have been allocated since the last collection cycle. Smaller heap sizes result in more frequent garbage collections than larger heap sizes. If the maximum heap size is reached, the garbage collector stops operating asynchronously, and user threads are forced to wait for collection cycles to complete. This situation has a significant negative impact on performance. A maximum heap size of 0 (*NOMAX) assures that garbage collection operates asynchronously at all times. For more information about tuning garbage collection with the JVM heap settings, see "Java virtual machine tuning parameters" on page 16.

The garbage collection gauge

You can use garbage collection to evaluate application performance health. Monitoring garbage collection when the server is under a fixed workload can help you determine if the application is creating several short-lived objects and can detect the presence of memory leaks.

You can monitor garbage collection statistics with any of these tools:
- The -verbosegc JVM configuration setting
  If you specify this setting, garbage collection generates verbose output.

  Note: The -verbosegc format is not standardized between different JVMs or release levels.
- The Dump Java Virtual Machine (DMPJVM) command
  This command dumps JVM information for a specified job.
- Heap Analysis Tools for Java(TM)

  This tool is a component of the iDoctor for iSeries suite of performance monitoring tools. The Heap Analysis Tools component performs Java application heap analysis and object create profiling (size and identification) over time. This tool is sometimes called Java Watcher or Heap Analyzer. For more information, about iDoctor for iSeries, see iDoctor for iSeries.
- Performance Explorer (PEX)

  You can use a Performance Explorer (PEX) trace to determine how much CPU is being used by the garbage collector. For detailed instructions, see Tuning Garbage Collection for Java(TM) and WebSphere on iSeries.

To obtain meaningful statistics, run the application under a fixed workload until the application state is steady. It usually takes several minutes to reach a steady state.

Detecting large numbers of short-lived objects

You can also use these tools to monitor JVM object creation:
- The DMPJVM (Dump Java Virtual Machine) command
  The DMPJVM command dumps information about the JVM for a specified job.
- The ANZJVM (Analyze Java Virtual Machine) command
  The ANZJVM command collects information about the Java Virtual Machine (JVM) for a specified job. This command is available in i5/OS V5R2 and later.
- The Performance Trace Data Visualizer (PTDV)

The best result for the average time between garbage collections is at least 5 to 6 times the average duration of a single garbage collection cycle. If the average time is shorter, the application is spending more than 15% of its time in garbage collection.

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

Detecting memory leaks

Memory leaks in the Java language are a significant contributor to garbage collection bottlenecks. Memory leaks are more damaging than memory overuse, because a memory leak ultimately leads to system instability. Over time, there is typically an increase in paging and garbage collection times. Garbage collection times increase until the heap is too large to fit into memory, paging rates increase, and eventually garbage collections are forced into synchronous mode. As a result, threads that are waiting for memory allocation are stopped. From a client's point of view, the application stops processing requests. Clients might also receive java.lang.OutOfMemoryError exceptions.

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 perform poorly after deployment in the production environment. This is especially true for leaking applications where the high workload accelerates the magnification of the leakage and the heap size grows too large for the garbage collector to manage.

Memory leak testing

The goal of memory leak testing is to magnify numbers. Memory leaks are measured in terms of the amount of bytes or kilobytes that garbage collection cannot collect. 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 contains important conclusions about memory leaks:

- **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 can lead to false alarms. 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.
- **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 unreleased 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.

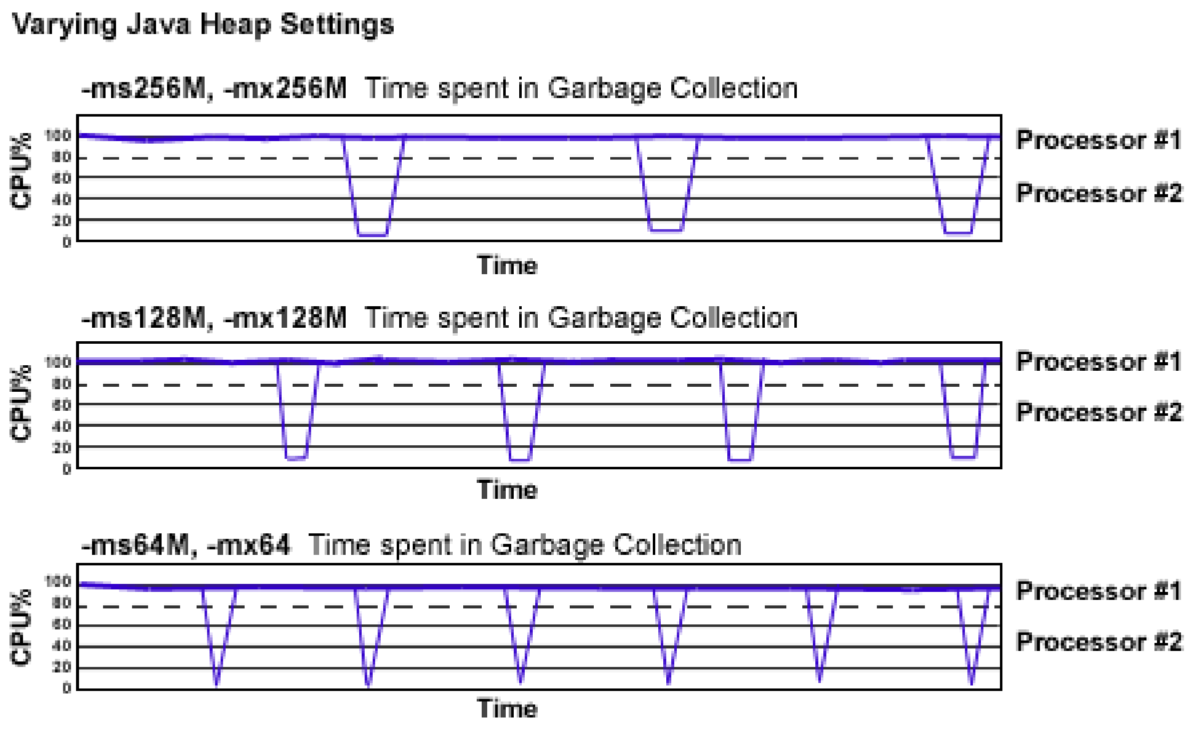
You can use these tools to detect memory leaks:
- The DMPJVM (Dump Java Virtual Machine) command
  The DMPJVM command dumps information about the JVM for a specified job.
- The ANZJVM (Analyze Java Virtual Machine) command
  The ANZJVM command collects information about the Java Virtual Machine (JVM) for a specified job. This command is available in i5/OS V5R2 and later.
- Heap Analysis Tools for Java(TM)

  This tool is a component of the iDoctor for iSeries suite of performance monitoring tools. The Heap Analysis Tools component performs Java application heap analysis and object create profiling (size and identification) over time. This tool is sometimes called Java Watcher or Heap Analyzer.

For best results, follow these guidelines:
- Use the tools to take a series of readings of the number of objects in the heap. Allowing at least 10 to 20 garbage collection cycles between each reading. You can compare the results of each reading to see if there are classes with a monotonically increasing count of objects.
- Repeat experiments with increasing duration, like 1000, 2000, and 4000-page requests. If the application uses several objects that are larger than 64KB, the total heap size may decrease after a garbage collection cycle. The JVM reuses the heap space for smaller objects, and only releases that space after long periods of idle activity. If the heap size continually increases and never reaches a steady state, there might be a memory leak.
- Look at the difference between the number of objects allocated and the number of objects freed. If the gap between the two increases over time, there is a memory leak. In most cases, determining object counts by class is the most useful way to detect leaks with the i5/OS JVM.

Java heap parameters

The Java heap parameters also influence the behavior of garbage collection. Because a large heap takes longer to fill, the application runs longer before a garbage collection occurs. For more information about heap settings, see "Java virtual machine tuning parameters" on page 16.
- **Initial heap size**
  When tuning a production system where the working set size of the Java application is not understood, it is recommended that you set the initial heap size to 96MB per processor. The total heap size in an i5/OS JVM can be approximated as the sum of the amount of live (in use) heap space at the end of the last garbage collection plus the initial heap size.



The illustration represents three CPU profiles, each running a fixed workload with a varying initial Java heap size. In the middle profile, the initial size is set to 128MB. Four garbage collections occur. The total time in garbage collection is about 15% of the total run. When the initial heap size is doubled to 256MB, 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 64MB 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.

This example shows that the total time in garbage collection is approximately the same in all cases. However, in most cases, setting a smaller initial heap size results in more total time spent in garbage collection, especially if the initial heap size is small compared to the pace of object allocation. If the initial heap size is too small the garbage collector runs almost continuously.

Run a series of test experiments that vary the initial Java heap settings. For example, run experiments with 128MB, 192MB, 256MB, and 320MB. During each experiment, monitor the total memory usage. When all of the runs are finished, compare these 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.

Note that unlike other JVM implementations, a large amount of heap free space is not generally a concern for the i5/OS JVM.

- **Maximum heap size**

  The maximum heap size can affect application performance. This value specifies the maximum amount of object space the garbage collected heap can consume. If the maximum heap size is too small, performance might degrade significantly, or the application might receive out of memory errors when the maximum heap size is reached. Due to the complexity of determining a correct value for the maximum heap size, a value of 0 (meaning there is no size limit) is recommended unless an absolute limit on the object space for the garbage collected heap size is required.

  In a situation where an absolute limit for the garbage collected heap is required, the value specified should be large enough so that performance is not negatively affected. To determine an appropriate value, run your application under a heavy load with a maximum heap value of 0. Determine the maximum size of the garbage collected heap for the JVM using DMPJVM or iDoctor. The smallest acceptable value for the maximum heap size is 125 percent of the garbage collected heap size. This value is a reasonable estimate for your garbage collected heap working set size. You can specify a larger value for the maximum heap size without affecting performance, and it is recommended that you set the largest possible value based on the resource restrictions of the JVM or the limitations of the system configuration.

After you determine an appropriate value for the maximum heap size, you might need to set up or adjust the pool in which the JVM runs. By default, WebSphere Application Server - Express jobs run in the base system pool (storage pool 2 as shown by WRKSYSSTS), but you can specify a different pool. The maximum heap size should not be set larger than 125 percent of the size of the pool in which the JVM is running. It is recommended that you run the JVM in its own memory pool with the memory permanently assigned to that pool, if possible.

If the performance adjuster is set to adjust the memory pools (that is, the system value QPFRADJ is set to a value other than 0), it is recommended that you specify a minimum size for the pool using WRKSHRPOOL. The minimum size should be approximately equal to your garbage collected heap working set size. Setting a correct maximum heap size and properly configuring the memory pool can prevent a JVM with a memory leak from consuming system resources, but still offers excellent performance.

When a JVM must run in a shared pool, it is more difficult to determine an appropriate value for the maximum heap size. Other jobs running in the pool can cause the garbage collected heap pages to be aged out of the pool. If the garbage collected heap pages are aged out of the pool, the garbage collector must fault the pages back into the pool on the next garbage collection cycle because it needs to access all of the pages in the garbage collected heap. Because the i5/OS JVM does not stop all of the JVM threads to clean the heap, excessive page faulting causes the garbage collector to slow down and the garbage collected heap to grow. Instead the size of the heap is increased and threads continue to run.

This heap growth is an artificial inflation of the garbage collected heap working set size, and must be considered if you want to specify a maximum heap value. When a small amount of artificial inflation occurs, the garbage collector reduces the size of the heap over time if the space remains unused and the activity in the pool returns to a steady state. However, in a shared pool, you might experience problems if the maximum heap size is set incorrectly

– If the maximum heap size is too small, artificial inflation can result in severe performance degradation or system failure if the JVM throws an out of memory error.
– If the maximum heap size is set too large the garbage collector might reach a point where it is unable to recover the artificial inflation of the garbage collected heap. In this case, performance is also negatively affected. A value that is too large might not be able to prevent a JVM failure, but it can prevent a run-away JVM from consuming excessive amounts of system resources.

If you want to determine the proper value for the maximum heap size, you must run multiple tests, because the appropriate value is different for each configuration or workload combination. If you want to prevent a run-away JVM, set the maximum heap size larger than you expect the heap to grow, but not so large that it affects the performance of the rest of the machine.

If you must set the maximum heap size to guarantee that the heap size does not exceed a given level, specify an initial heap size that is 80-90% smaller than the maximum heap size.

Web server tuning parameters

WebSphere Application Server - Express provides plug-ins for several Web server brands and versions. If you are running your Web server on a non-iSeries platform, see the product documentation for performance tuning information. For additional information, refer to Chapter 6 of the Performance Capabilities Reference Manual. This manual is available in the Performance Management Resource Library.

The IBM HTTP Server (powered by Apache) is a multi-process, multi-threaded server. For IBM HTTP Server for i5/OS, you can adjust these settings:

- **Access logs**
  - Description: The access logs record all incoming HTTP requests. Logging can degrade performance. On iSeries, overhead is minimized, because logging occurs in a separate process from the Web server function.
  - How to view or set:
    1. Open the IBM HTTP Server httpd.conf file, located in the /QIBM/ProdData/HTTPA/conf directory.
    2. Search for lines with the text CustomLog.
    3. To enable a custom access log, remove the hash mark (#) at the beginning of the line.
    4. Save and close the httpd.conf file.
    5. Stop and restart the IBM HTTP Server.
  - Default value: By default, the access log is disabled.
  - Recommended value: Do not enable the access logs.
- **ThreadsPerChild**
  - Description: This directive specifies the maximum number of concurrent client requests that the server processes at any time. The Web server uses one thread for each request that it processes. This value does not represent the number of active clients.
  - How to view or set: Edit or view the ThreadsPerChild directive in the IBM HTTP Server httpd.conf file.
  - Default value: 40
  - Recommended value: It is recommended that you use the default value, and only increase this value if necessary.
- **ListenBackLog**
  - Description: This parameter sets the length of a pending connections queue. When several clients request connections to the IBM HTTP Server, and all threads are in use, a queue exists to hold additional client requests. However, if you use the default Fast Response Cache Accelerator (FRCA) feature, the ListenBackLog directive is not used, because FRCA uses its own internal queue.
  - How to view or set: For non-FRCA: Edit or view the ListenBackLog directive in the IBM HTTP Server httpd.conf file.
  - Default value: For IBM HTTP Server 1.3.26: 1024 with FRCA enabled, 511 with FRCA disabled
  - Recommended value: Use the default value.

Database tuning parameters

For tuning information for DB2 UDB for iSeries, see these resources:

- Query performance and query optimization in the *Database* topic.
- Chapter 4 of the Performance Capabilities Reference. Links to several editions of the Performance Capabilities Reference are listed in the Performance Management Resource Library.

You might also want to adjust the QSQSRVR prestart job settings for your system. On iSeries, QSQSRVR jobs process Java database access (JDBC). By default, five QSQSRVR jobs are initially active. When fewer than two QSQSRVR jobs are unused, two more jobs are created. An application that establishes a large number of database connections over a short period of time may be able to create connections more quickly if these values are increased. To increase the initial number of jobs, threshold, and additional number of jobs values, run this command on an CL command line:

```
CHGPJE SBSD(QSYS/QSYSWRK) PGM(QSYS/QSQSRVR)
```

Do not start more QSQSRVR jobs than your application requires, because there is some overhead associated with having QSQSRVR jobs active, even if they are not being used.

If you use a different database, refer to that product's documentation.

TCP/IP buffer sizes

WebSphere Application Server - Express 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.

Flow control can consume a significant amount of CPU time and result in additional network latency as a result of data transfer interruptions. It is recommended that you increase buffer sizes to avoid flow control under normal operating conditions. A larger buffer size reduces the potential for flow control to occur, and results in improved CPU utilization. However, a large buffer size can have a negative effect on performance in some cases. If the TCP/IP buffers are too large and applications are not processing data fast enough, paging can increase. The goal is to specify a value large enough to avoid flow control, but not so large that the buffer accumulates more data than the system can process.

The default buffer size is 8KB. The maximum size is 8MB. The optimal buffer size depends on several network environment factors including types of switches and systems, acknowledgment timing, error rates, network topology, memory size, and data transfer size. When data transfer size is extremely large, you might want to set the buffer sizes up to the maximum value to improve throughput, reduce the occurrence of flow control, and reduce CPU cost.

Buffer sizes for the socket connections between the Web server and WebSphere Application Server - Express are set at 64KB. In most cases this value is adequate.

Flow control can be an issue when an application uses either the IBM Developer Kit for Java$^{(TM)}$ JDBC driver or the IBM Toolbox for Java JDBC driver to access a remote database. If the data transfers are large, flow control can consume a large amount of CPU time. If you use the IBM Toolbox for Java JDBC driver, you can use custom properties to configure the buffer sizes for each data source. It is recommended that you specify large buffer sizes, such as 1MB.

Some system-wide settings can override the default 8KB buffer size for sockets. With some applications, such as WebSphere Commerce Suite, a buffer size of 180KB reduces flow control and typically does not increase paging. The optimal value is dependent on specific system characteristics. You might need to try several values before you determine the ideal buffer size for your system. To change the system wide value follow these steps:

1. Run the Change TCP/IP Attributes (CHGTCPA) command on an CL command line.
2. On the **Change TCP/IP Attributes** display, press **F4**. The buffer sizes are displayed as the TCP receive and send buffer size.
3. Specify new values.
4. Save your changes.
5. Recycle TCP/IP.
6. Monitor CPU and paging rates to determine if they are within recommended system guidelines.

Repeat this process until you determine the ideal buffer size.

For more information about TCP/IP performance, see Chapter 5 of the Performance Capabilities Reference. Links to several editions of the Performance Capabilities Reference are listed in the

Performance Management Resource Library.

Application assembly performance checklist

Application assembly tools are used to build J2EE components and modules into J2EE applications. Application assembly consists of defining application components and their attributes including enterprise beans, servlets and resource references. Many of these application configuration settings and attributes play an important role in the run-time performance of the deployed application. Use this information as a check list of important parameters and advice for finding optimal settings:

- Web modules assembly settings
  - Distributable
  - Reload interval
  - Reload enabled
- Web component settings
  - Load on startup

You can also use the JSP pre-touch tool to enhance application server performance. This tool causes all JSPs to be compiled when your application server starts. For more information, see Pre-touch tool for compiling and loading JSP files in the *Application development* topic.

Appendix. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

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

```
IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY  10504-1785
U.S.A.
```

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

```
IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan
```

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

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:

Programming Interface Information

This WebSphere Application Server - Express publication documents intended Programming Interfaces that allow the customer to write programs to obtain the services of IBM i5/OS.

Trademarks

Terms and conditions

Printed in USA