WebSphere Business Integration V6.0.2 Performance Tuning

Learn valuable tips for tuning

Get the latest best practices

Try the example settings

IBM WebSphere Process Server, WebSphere Enterprise Service Bus, WebSphere Adapters, and WebSphere Business Monitor Performance teams

ibm.com/redbooks
First Edition (June 2007)

This edition applies to WebSphere Process Server 6.0.2, WebSphere Enterprise Service Bus 6.0.2, WebSphere Adapters 6.0.2, and WebSphere Business Monitor 6.0.2.

This document created or updated on June 7, 2007.
Chapter 1. Performance tuning and configuration

1.1 Tuning checklist ................................................. 2
1.2 Common tuning parameters ................................. 3
  1.2.1 Disable tracing and monitoring when possible .... 3
  1.2.2 Move databases off of the default Cloudscape ...... 3
  1.2.3 Set Java heap size appropriately ................. 4
1.3 WebSphere Process Server .................................. 4
  1.3.1 Do not run a production server in development mode .... 4
  1.3.2 Move the messaging engine data stores to a high performance DBMS ...... 4
  1.3.3 Configure threads for messaging and in work managers .... 7
  1.3.4 Configure WebSphere Process Server for clustering .... 7
  1.3.5 Disable validation in CEI emitter .................. 9
1.4 WebSphere Enterprise Service Bus ................... 9
  1.4.1 Configure the Web container thread pool settings .... 9
  1.4.2 Optimize the activation specification (JMS) .......... 9
  1.4.3 Optimize the QueueConnectionFactory (JMS) .... 10
  1.4.4 Optimize the listener pool configuration (JMS or MQ) ..... 10
  1.4.5 Use HTTP persistent connections .................... 10
  1.4.6 Configure data buffer sizes ....................... 11
  1.4.7 Tune the database if using persistent messaging .... 11
  1.4.8 Disable event distribution for CEI ................. 11
1.5 WebSphere adapters ......................................... 12
  1.5.1 Configure poll period and poll quantity ............. 12
  1.5.2 Configure the application server thread pool ...... 13
  1.5.3 Configure the work manager thread pool .......... 13
1.6 WebSphere Business Monitor ............................ 14
  1.6.1 Configure CEI .................................. 14
  1.6.2 Set the event processing batch size ............. 15
1.7 Database: General ............................................ 16
  1.7.1 Place database log files on a fast disk subsystem .... 16
  1.7.2 Size database cross-referencing tables correctly ... 16
  1.7.3 Place logs on a separate device from table spaces .... 17
1.8 Database: DB2-specific ..................................... 17
  1.8.1 Maintain current indexes on tables ............... 17
  1.8.2 Update catalog statistics .......................... 17
  1.8.3 Set buffer pool sizes correctly ............. 18
1.9 Java ......................................................... 19
  1.9.1 Set the heap and nursery size to handle garbage collection efficiently .... 19
  1.9.2 Set AIX threading parameters .................... 23
  1.9.3 Reduce or increase heap size if OutOfMemory errors occur .... 23
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 give 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.

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.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.
Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

- Redbooks (logo)®
- AIX®
- Cloudscape™
- DB2 Universal Database™
- IBM®
- MQSeries®
- Redbooks®
- WebSphere®

The following terms are trademarks of other companies:

mySAP.com, mySAP, SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

Oracle, JD Edwards, PeopleSoft, Siebel, and TopLink are registered trademarks of Oracle Corporation and/or its affiliates.

Enterprise JavaBeans, EJB, Java, Java HotSpot, JavaBeans, JDBC, JVM, J2EE, Solaris, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, and the Windows logo 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.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

This IBM® Redpaper publication was produced by the IBM WebSphere® Process Server, WebSphere Enterprise Service Bus, WebSphere Adapters, and WebSphere Business Monitor performance teams in Austin, Texas; Böblingen, Germany; and Hursley, England. It provides performance tuning tips and best practices based on the performance teams’ experience in using the following products:

- WebSphere Process Server 6.0.2
- WebSphere Enterprise Service Bus (WebSphere ESB) 6.0.2
- WebSphere Adapters 6.0.2
- WebSphere Business Monitor 6.0.2

These products represent an integrated development and runtime environment based on a key set of service-oriented architecture (SOA) technologies: Service Component Architecture (SCA), Service Data Object (SDO), and Business Process Execution Language for Web Services (BPEL). These technologies in turn build on the core capabilities of WebSphere Application Server 6.0.2, including the Java™ Connector Architecture (JCA) V1.5 on which WebSphere Adapters v6.0.2 is based. WebSphere Business Monitor offers the ability to monitor business processes in real time, providing a visual display of business process status, together with alerts and notifications to key users that enables continuous improvement of business processes.

If you are either considering or are in the early stages of implementing a solution that incorporates these products, this document should prove to be a useful reference. It describes best practices during application development and deployment and provides a starting point for setup, tuning, and configuration information. It contains a useful introduction to many of the issues that affect these products’ performance and can act as a guide for making rational first choices in terms of configuration and performance settings.

Finally, all of these products build upon the capabilities of the WebSphere Application Server infrastructure. Therefore, solutions that are based on these products will also benefit from existing tuning, configuration, and best practices information for WebSphere Application Server documented in “Related publications” on page 41. You are encouraged to use the information in this paper in conjunction with these references.
The team that wrote this paper

This document was produced by the following members of the IBM WebSphere Process Server, WebSphere Enterprise Service Bus, WebSphere Adapters, and WebSphere Business Monitor performance teams in Austin, Böblingen, and Hursley.

Phani Achanta
Rajiv Arora
Mike Collins
Gudrun Eckl-Regenhardt
Ivan Eisen
Steve Garde
Jesse Gordon
Jonas Grundler
Weiming Gu
Steve Heuer
Ben Hoflich
Thomas Muehlfriedel
Gerhard Pfau
Sefika Prcic
Chris Richardson
IBM WebSphere Process Server performance team

Paul Harris
Sam Massey
Rachel Norris
IBM WebSphere Enterprise Service Bus performance team

Jimmy DeWitt
Riaz Hussain
IBM WebSphere Adapters performance team

Bill Alexander
Walter Fraser
Kean Kuiper
IBM WebSphere Business Monitor performance team

Become a published author

Join us for a two- to six-week residency program! Help write a book dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:
ibm.com/redbooks/residencies.html
Comments welcome

Your comments are important to us!

We want our papers to be as helpful as possible. Send us your comments about this paper or other IBM Redbooks® publications in one of the following ways:

► Use the online Contact us review Redbooks form found at:
  ibm.com/redbooks

► Send your comments in an e-mail to:
  redbooks@us.ibm.com

► Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400
Performance tuning and configuration

To optimize performance, it is usually necessary to configure the system differently than the default settings. This chapter lists several areas to consider during system tuning. These areas include tuning the WebSphere Process Server, WebSphere Enterprise Service Bus (WebSphere ESB), WebSphere Integration Developer, WebSphere Adapters and Monitor products, and other products in the system. As an example, WebSphere Process Server supports different database managers. The documentation for each of these products contains a wealth of information regarding performance, capacity planning, and configuration. This documentation likely offers the best guidance for performance considerations in a variety of operational environments. Assuming that all these issues have been addressed from the perspective of the product, additional levels of performance implications are introduced at the interface between these products and the products covered in this paper.

A number of configuration parameters are available to the system administrator. While this chapter identifies several specific parameters that have been observed to affect performance, it does not address all available parameters. For a complete list of configuration parameters and possible settings, see the relevant product documentation.

This chapter begins with a tuning checklist that enumerates the major components and their associated tuning concepts. The subsections that follow address each in more detail. First we describe performance-related design concepts. Then we discuss the tuning parameters and their suggested settings (where appropriate). Finally we provide suggested ways to determine potential settings for a particular configuration.

Note: While there is no guarantee that stepping through this checklist will immediately provide acceptable performance, it is likely that degraded performance can be expected if any of these parameters are set incorrectly.

The last section of this document contains references to related documentation that may prove valuable when tuning a particular configuration.
### 1.1 Tuning checklist

Table 1-1 contains a checklist of basic tuning actions for each product. We discuss these actions in more detail in later sections.

<table>
<thead>
<tr>
<th>Product</th>
<th>Tuning action</th>
</tr>
</thead>
</table>
| **Common** | ▶ Disable tracing and monitoring when possible.  
▶ Move databases off of the default Cloudscape™ to a high performance database management system (DBMS) (WebSphere Process Server, WebSphere Enterprise Service Bus, and WebSphere Adapters).  
▶ Set Java heap size appropriately (see the Java section of this tuning checklist).  
▶ Do not enable security, where practical.  
▶ Use the appropriate hardware configuration for performance measurement. For example, ThinkPads and desktops are not appropriate for realistic performance evaluations. |
| **WebSphere Process Server** | ▶ Do not run a production server in development mode.  
▶ Move the default messaging provider data stores to a high performance DBMS.  
▶ Configure threads for messaging and in work managers.  
▶ Configure WebSphere Process Server for clustering (where applicable).  
▶ Disable validation in the Common Even Interface (CEI) emitter.  
▶ Do not use the Unit Test Environment (UTE) for performance measurements. |
| **WebSphere Enterprise Service Bus** | ▶ Configure Web container thread pool maximum and minimum sizes.  
▶ Optimize the activation specification (Java Message Service (JMS)).  
▶ Optimize Queue Connection factory (JMS).  
▶ Optimize the Listenet Port configuration (JMS and MQ).  
▶ Configure HTTP connection KeepAlive.  
▶ Configure data buffer sizes.  
▶ Tune the database, if using persistent messaging.  
▶ Disable event distribution for CEI (CEI mediations). |
| **WebSphere adapters** | ▶ Configure pollPeriod and pollQuantity.  
▶ Configure the application server thread pool.  
▶ Configure the work manager thread pool. |
| **WebSphere Business Monitor** | ▶ Configure the CEI.  
▶ Set the event processing batch size. |
| **Database: General** | ▶ Place database tablespaces and logs on a fast disk subsystem.  
▶ Size the database cross-referencing tables correctly.  
▶ Place logs on a separate device from table spaces. |
| **Database: DB2-specific** | ▶ Maintain current indexes on tables.  
▶ Update the catalog statistics.  
▶ Set the buffer pool size correctly. |
| **Java** | ▶ Set the heap/nursery sizes to manage memory efficiently.  
▶ Set AIX® threading parameters.  
▶ Reduce or increase heap size if java.lang.OutOfMemory occurs. |
1.2 Common tuning parameters

The following sections cover tuning options that are common to all products that we discuss in this report, except where noted otherwise.

1.2.1 Disable tracing and monitoring when possible

The ability to configure tracing and monitoring at different levels for a variety of system components has proven to be extremely valuable during periods of system analysis or debugging. The WebSphere Business Integration product set provides rich monitoring capabilities, both in terms of business monitoring via the Common Event Interface and audit logging, and system performance monitoring via Performance Monitoring Infrastructure (PMI) and Application Response Measurement (ARM). While these capabilities provide insight into the performance of the running solution, these features can degrade overall system performance and throughput.

Important: We recommend that you use tracing and monitoring judiciously, and when possible, turn it off entirely to ensure optimal performance.

Most tracing and monitoring is controlled via the WebSphere Application Server administrative console. Validate that the appropriate level of tracing and monitoring is set for PMI monitoring, logging, and tracing via the administrative console.

► To disable tracing, select Troubleshooting → Logs and Trace → server_name → Change Log Detail Levels and set both the Configuration and Runtime to *=all=disabled.

► To change the monitoring level, select Monitoring and Tuning → Performance Monitoring Infrastructure → server_name and select none.

In addition, use the administrative console to validate that the Audit logging and Common Event Infrastructure logging check boxes are disabled in the Business Process container, unless these capabilities are required for business reasons.

WebSphere Integration Developer is also used to control event monitoring. Check the Event Monitor tab for your components and business processes to ensure that event monitoring is applied judiciously.

1.2.2 Move databases off of the default Cloudscape

WebSphere Process Server, WebSphere Enterprise Service Bus, and WebSphere Adapters are packaged with the Cloudscape database, an open source database designed for ease-of-use and platform neutrality.

When performance and reliability are critically important, use an industrial strength database, such as IBM DB2®, for any performance measurement or production installation. Examples of databases that can be moved to DB2 include the BPE database, relationship databases, bus messaging engine data stores, adapter event delivery tables, and any adapter specific databases.

The conversion requires the administrator to create new Java Database Connectivity (JDBC™) providers in the administrative console under Resources → JDBC Providers. When the provider has been created, a data source can be added to connect to a database.
A key data source property is Maximum Connections under Connection pools additional properties. This should be set high enough so worker threads do not have to wait for a connection from the pool.

1.2.3 Set Java heap size appropriately

The size of the Java heap can greatly influence several aspects of product performance and robustness. You can find the initial heap size and maximum heap size settings for an application server Java virtual machine (JVM™) by selecting Servers → Application servers → server_name → Server Infrastructure → Java and Process Management → Process Definition → Additional Properties → Java Virtual Machine.

See 1.9.1, “Set the heap and nursery size to handle garbage collection efficiently” on page 19, and 1.9.3, “Reduce or increase heap size if OutOfMemory errors occur” on page 23, for further information about determining appropriate values for these settings.

1.3 WebSphere Process Server

In this section, we discuss performance tuning tips for WebSphere Process Server.

1.3.1 Do not run a production server in development mode

WebSphere application servers can be run in development mode. Development mode reduces startup time for the server that is using JVM settings to disable bytecode verification and reduce JIT compilation costs. This setting should not be used on production servers.

You can find the setting in the administrative console by selecting Servers → Application Servers → server_name → Configuration.

1.3.2 Move the messaging engine data stores to a high performance DBMS

The messaging engine data stores are automatically created during profile creation on Cloudscape without allowing any user choice. The following example demonstrates one method to create alternative data stores for the messaging engines on DB2.

After the profile creation wizard has finished and Business Process Choreographer is configured, the system should contain four buses, with one messaging engine each. Table 1-2 shows the buses in WebSphere Process Server installed on machine “box01”.

<table>
<thead>
<tr>
<th>Bus</th>
<th>Messaging engine</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCA.SYSTEM.box01Node01Cell.Bus</td>
<td>box01-server1.SCA.SYSTEM.box01Node01Cell.Bus</td>
</tr>
<tr>
<td>SCA.APPLICATION.box01Node01Cell.Bus</td>
<td>box01-server1.SCA.APPLICATION.box01Node01Cell.Bus</td>
</tr>
<tr>
<td>CommonEventInfrastructure_Bus</td>
<td>box01-server1.CommonEventInfrastructure_Bus</td>
</tr>
<tr>
<td>BPC.box01Node01Cell.Bus</td>
<td>box01-server1.BPC.box01Node01Cell.Bus</td>
</tr>
</tbody>
</table>

Note: The node and cell names are the default.
Each of these messaging engines is configured to use a data store in Cloudscape. Each data store is located in its own database. When considering DB2, this is not optimal from an administrative point of view. There are already many databases in the system; adding four more databases increases the maintenance and tuning effort substantially. The solution proposed here uses a single DB2 database for all four data stores. The individual data stores or tables are completely separate and each messaging engine acquires an exclusive lock on its set of tables during startup. Each messaging engine uses a unique schema name to identify its set of tables.

**Setting up the data stores for the messaging engines**

For information about data stores, see “Setting up the data store for a messaging engine” on the Web at:


**Create the DB2 database and load the data store schemas**

Instead of having a DB2 database per messaging engine, we put all messaging engine data stores into the same database using several schemas (Table 1-3) to separate them.

<table>
<thead>
<tr>
<th>Schema</th>
<th>Messaging engine</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCASYS</td>
<td>box01-server1.SCA.SYSTEM.box01Node01Cell.Bus</td>
</tr>
<tr>
<td>SCAAPP</td>
<td>box01-server1.SCA.APPLICATION.box01Node01Cell.Bus</td>
</tr>
<tr>
<td>CEIMSG</td>
<td>box01-server1.CommonEventInfrastructure_Bus</td>
</tr>
<tr>
<td>BPCMSG</td>
<td>box01-server1.BPC.box01Node01Cell.Bus</td>
</tr>
</tbody>
</table>

To create one schema definition for each messaging engine:

1. Create one schema definition for each messaging engine by entering the following command on Microsoft® Windows®:

   ```
   app_server_root\bin\sibDDLGenerator.bat -system db2 -version 8.1 -platform windows -statementend ; -schema BPCMSG -user user >createSIBSchema_BPCMSG.ddl
   ```

   Here, `app_server_root` represents the WebSphere Process Server Installation directory, and `user` represents the user name.

2. Repeat step 1 for each schema or messaging engine.

3. To distribute the database across several disks, edit the created schema definitions and put each table in a table space named after the schema that is used. For example, SCAAPP becomes SCANODE_TS, CEIMSG becomes CEIMSG_TS, and so on. The schema definition should look like Example 1-1 after editing.

**Example 1-1  Commands to create the schema and table**

```
CREATE SCHEMA CEIMSG;
CREATE TABLE CEIMSG.SIBOWNER (ME_UUID VARCHAR(16), INC_UUID VARCHAR(16), VERSION INTEGER, MIGRATION_VERSION INTEGER) IN CEIMSG_T8;
```

```
CLASSID INTEGER NOT NULL,
URI VARCHAR(2048) NOT NULL,
PRIMARY KEY(CLASSID)
) IN CEIMSG_TB;
...

It is possible to provide separate table spaces for the various tables here. Optimal distributions depend on application structure and load characteristics. In the example, one table space per data store was used.

4. After creating all schema definitions and defined table spaces for the tables, create a database named SIB.

5. Create the table spaces and distribute the containers across available disks as fitting by entering the following command for a system managed table space:

`DB2 CREATE TABLESPACE CEIMSG_TB MANAGED BY SYSTEM USING( 'path\CEIMSG_TB' )`

6. Place the database log on a separate disk if possible.

7. Create the schema of the database by loading the four schema definitions into the database.

**Create the data sources for the messaging engines**

Create a data source for each messaging engine, and configure each messaging engine to use the new data store using the administrative console.

Table 1-4 shows the default state.

<table>
<thead>
<tr>
<th>Messaging engine</th>
<th>JDBC provider</th>
</tr>
</thead>
<tbody>
<tr>
<td>box01-server1.SCA.SYSTEM.box01Node01Cell.Bus Cloudscape JDBC Provider (XA)</td>
<td></td>
</tr>
<tr>
<td>box01-server1.SCA.APPLICATION.box01Node01Cell.Bus Cloudscape JDBC Provider</td>
<td></td>
</tr>
<tr>
<td>box01-server1.CommonEventInfrastructure_Bus Cloudscape JDBC Provider</td>
<td></td>
</tr>
<tr>
<td>box01-server1.BPC.box01Node01Cell.Bus Cloudscape JDBC Provider</td>
<td></td>
</tr>
</tbody>
</table>

To create the data sources:

1. Create a JDBC provider using the DB2 Universal JDBC Driver Provider for the non-XA data sources if you do not already have one. The XA DB2 JDBC Driver Provider should exist if BPC was configured correctly for DB2.

2. Create four new JDBC data sources: one for CEI as an XA data source and the remaining three as single-phase commit (non-XA) data sources.

Table 1-5 provides new names and values for the data sources.

<table>
<thead>
<tr>
<th>Name of data source</th>
<th>JNDI name</th>
<th>Type of JDBC provider</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEIMSG_sib</td>
<td>jdbc/sib/CEIMSG</td>
<td>DB2 Universal (XA)</td>
</tr>
<tr>
<td>SCAAPP_sib</td>
<td>jdbc/sib/SCAAPPLICATION</td>
<td>DB2 Universal</td>
</tr>
<tr>
<td>SCASYSTEM_sib</td>
<td>jdbc/sib/SCASYSTEM</td>
<td>DB2 Universal</td>
</tr>
<tr>
<td>BPCMSG_sib</td>
<td>jdbc/sib/BPCMSG</td>
<td>DB2 Universal</td>
</tr>
</tbody>
</table>
When creating each data source, set the following options (in no particular order):

- Deselect the check box labeled **Use this Data Source in container managed persistence (CMP)**.
- Set a component-managed authentication alias.
- Set the database name to the name used for the database created earlier for messaging (SIB).
- Select a driver type of 2 or 4; both should work. Per DB2 recommendations, you should use the JDBC Universal Driver Type 2 connectivity to access local databases and Type 4 connectivity to access remote databases. A driver type of 4 requires a host name and valid port to be set on the database.

**Change the data stores of the messaging engines**

Use the administrative console to change the data stores of the messaging engines:

1. For each messaging engine, select **Service Integration → Buses**, and change the data stores for each messaging engine on each bus.
2. Type the new JNDI and schema name for each data store. Deselect the **Create Tables** check box since the tables have been created already.
   
The server immediately restarts the messaging engine. The SystemOut log shows the results of the change and if the messaging engine comes up successfully.
3. Restart the server and validate that all systems come up using the updated configuration.

The last remaining task is to tune the database using the DB2 Configuration Wizard as a starting point and collect statistics after the tables are populated. Also check the relevant database parameters such as the number of connections that have to satisfy the requirements of the four messaging engines.

### 1.3.3 Configure threads for messaging and in work managers

The **Platform Messaging Component SPI Resource Adapter** is created as part of an application installation. It provides an enterprise information system (EIS) with the ability to communicate with message-driven beans (MDBs) configured on the server. Message processing for an application uses properties that are defined under an activation specification for this resource adapter.

The maxConcurrency custom property under the J2C activation specification for the resource adapter is used to specify the number of threads that are available to process messages by the application.

If a work manager is used, set the Maximum number of threads property to a value that is high enough to prevent the thread pool from running out of threads. You can set this value in the administrative console under **Asynchronous beans → work managers**.

One symptom of insufficient concurrency is CPU idleness. Vary the concurrency to achieve maximum CPU utilization and throughput.

### 1.3.4 Configure WebSphere Process Server for clustering

When tuning to achieve the best possible horizontal scalability through clustering, additional WebSphere Process Server configuration is required.
Configure the activation specification properties
Each Service Component Architecture (SCA) module defines an MDB and its corresponding activation specification. The default value for maxConcurrency of the SCA module MDB is 10 meaning that only up to 10 asynchronous SCA requests in the module can be processed concurrently. If the server CPU is not maxed out, it is sometimes caused by this setting being too low; it needs to be increased.

2. Select the SCA module activation specification.
3. Click J2C activation specification custom properties.
4. Change maxConcurrency to a higher value, for example 20.

Configure the ORB thread pool
The ORB thread pool configuration parameter is relevant if the cluster is driven by a driver node through the SCA synchronous binding.

Due to the interaction between synchronous SCA, workload management (WLM), and the Object Request Broker (ORB), the ORB thread pool size on the cluster nodes needs to be configured to maximize the clustering throughput. The rule of the thumb is to use the same number of ORB threads on all application nodes. You must also have the total number of ORB threads across all application nodes be the same as the number of driver threads on the driver node. For example, if the driver uses 120 concurrent threads, the ORB thread pool size on each application node on a 6-node cluster should be 20.

The default value of ORB thread is 50. To set this value, select Servers → Application servers → server_name → Container Services → ORB Service → Thread Pool. Then complete the Maximum Size field.

Configure the relationship and BPE data source connection pools
The Maximum Connections property of the relationship data source should be large enough to allow concurrent access to the database from all threads. For the clustered measurements in the test lab, this property was set to two times the largest value of the WebContainer, ORB, or the default thread pool size. The multiplier of 2 comes from the fact that each transaction in the test run could contain two relationship database accesses. If your application contains more or less relationship database access, you need to adjust the multiplier accordingly.

The Maximum Connections property of the BPE data source should be large enough to allow concurrent access to the database from all threads. It should be larger than the largest value of the WebContainer, ORB, or the default thread pool size.

To set maximum connections for a data source, select Resources → JDBC Providers → JDBC provider name → Data sources → Data source name → Connection pool properties. Then complete the Maximum connections field.

To find the Web container thread pool size, select Servers → Application servers → server_name → Thread Pools → WebContainer. The value is in the Maximum Size field.

You can find the maximum connections for the ORB service by selecting Servers → Application servers → server_name → Container Services → ORB Service → Thread Pool. The value is in the Maximum Size field.

To find the default thread pool size, select Servers → Application servers → server_name → Thread Pools → Default. The value is in the Maximum Size field.
Configure the IBM HTTP Server
On the driver node, set the `com.ibm.websphere.webservices.http.maxConnection` system property to 200. The default is 50. In the performance tests in a clustered environment, more than 50 driver threads were needed to max out the CPU on six cluster members.

In `httpd.conf` for the IBM HTTP Server, set `MaxSpareThreads` to 200. (The default is 75.) This setting enables more than 75 driver threads, which were needed for the clustered workload performance tests using Web services to run without errors.

Configure the JMSImport connection factory
The maximum connections property of the JMSImport Connection Factory (SAPAdpt.JMSImport_CF) connection pool should be large enough to provide a connection for every driver thread. The default is 10.

As an example, in internal performance testing, 50 driver threads were needed to max out the CPU on six cluster members. The maximum connections property was set to 50.

1.3.5 Disable validation in CEI emitter

The default behavior of Common Even Infrastructure (CEI) event emitters is to validate the events to ensure that all fields are correctly filled. This is unnecessary for events that are generated by the WebSphere Process Server system.

To disable event validation, add a custom property to the emitter factory profile:

1. Select `Resources → CEI Infrastructure Provider` (cell scope) → `Emitter Factory Profile` → `Default Common Event Infrastructure emitter` → `Custom Properties`.
2. Add a new property with a name of `validateEvent` and a value set to `false`.

1.4 WebSphere Enterprise Service Bus

In the following section, we discuss configuration options that are relevant to tuning WebSphere ESB. See 3.1, "WebSphere ESB settings" on page 38, for a list of the values that were used during the WebSphere Enterprise Service Bus performance tests.

1.4.1 Configure the Web container thread pool settings

Web container thread pool sizes and timeouts need to be set to optimum values. You can find these settings by selecting `Servers → Application servers` → `<server_name>` → `Additional Properties` → `Thread Pools` → `WebContainer`.

1.4.2 Optimize the activation specification (JMS)

Increase the maximum concurrent endpoints. This is the maximum number of endpoints to which messages are delivered concurrently. Increasing this number can improve performance as well as the number of threads that are in use at any one time. If message ordering must be retained across failed deliveries, set the maximum concurrent endpoints to 1.

You can find the maximum concurrent endpoints by selecting `Resources → JMS Providers` → `Default messaging` → `Activation specifications` → `JMS activation specification` → `activation_specification`.
1.4.3 Optimize the QueueConnectionFactory (JMS)

Increase the maximum connection pool size to allow for greater concurrency. It should be noted that if only one deployed application is obtaining connections from a particular connection factory, there would be no benefit in increasing the maximum queue connection pool size beyond the maximum concurrency as configured within the activation specification.

The maximum connection pool size specifies the maximum number of physical connections that you can create in this pool. These are the physical connections to the backend resource, for example a DB2 database. When this number is reached, no new physical connections are created, and the requester waits until a physical connection that is currently in use returns to the pool, or a ConnectionWaitTimeout exception is issued.

For example, if the Maximum Connections value is set to 5, and there are five physical connections in use, the pool manager waits for the amount of time specified in Connection Timeout for a physical connection to become free. If no connection is freed in the specified interval, a ConnectionWaitTimeout exception is issued.

If Maximum Connections is set to 0, the connection pool is allowed to grow infinitely. This also has the side effect of causing the Connection Timeout value to be ignored.

You can find the maximum connection pool size by selecting **Resources → JMS Providers → Default messaging → JMS queue connection factory → connection_factory → Additional properties → Connection pool properties**.

1.4.4 Optimize the listener pool configuration (JMS or MQ)

If MQ or MQ JMS bindings are used, a listener port is used to configure the delivery of inbound messages into the WebSphere ESB. The MaxSession property serves a similar purpose to the number of concurrent endpoints within activation specifications. Increase this value to allow more work to be concurrently processed. The queue connection factory (QCF) that is referenced from a listener port also contains a SessionPool with minimum and maximum pool sizes that may also need to be increased to support the desired concurrency within the listener port.

You can find the MaxSession parameter by selecting **Servers → Application servers → Server1 → Communications → Messaging → Messaging Listener Service → Listener Ports → listener port**.

**Note:** A listener port is created for you when using the MQ bindings.

You can find the maximum session pool size for the MQ QCF by selecting **Resources → JMS Providers → WebSphere MQ → WebSphere MQ queue connection factories → connection_factory → Additional properties → Session pools**.

1.4.5 Use HTTP persistent connections

Ensure that HTTP persistent connections are used in the HTTP transport channels and for Web service client requests. Persistent connection use is enabled by default, but you can check that your server is configured correctly by referring to the following sections in the WebSphere Application Server Information Center:
1.4.6 Configure data buffer sizes

The `sib.msgstore.discardableDataBufferSize` setting is the size in bytes of the data buffer that is used when processing best effort nonpersistent messages. The purpose of the discardable data buffer is to hold message data in memory because it is never written to the data store for this quality of service. Messages that are too large to fit into this buffer are discarded.

The `sib.msgstore.cachedDataBufferSize` setting is the size in bytes of the data buffer that is used when processing all messages other than best effort nonpersistent messages. The purpose of the cached data buffer is to optimize the performance by caching in memory data that might otherwise need to be read from the data store.

You can create the `sib.msgstore.discardableDataBufferSize` and `sib.msgstore.cachedDataBufferSize` settings by selecting Service Integration → Buses → bus_name → Messaging Engines → messaging_engine → Additional properties → Custom properties.

1.4.7 Tune the database if using persistent messaging

If you are using persistent messaging, the configuration of your database becomes important. Use a remote DB2 instance with a fast disk array as the database server. You may also find benefit in tuning the connection pooling and statement cache of the data source. See 1.7, “Database: General” on page 16, and 1.8, “Database: DB2-specific” on page 17, for further information about tuning DB2. Also note the relevant references in “Related publications” on page 41.

1.4.8 Disable event distribution for CEI

The event server that manages events can be configured to distribute events, log them to the event database, or do both. The mediation tests only require events to be logged to a database. Therefore, performance is improved by disabling event distribution. Since the event server may be used by other applications, it is important to check that none of the applications uses event monitoring before disabling event distribution.

You can disable event distribution by selecting Resources → Common event Infrastructure. Set the scope to Cell. Then select Event Server Profile and deselect EnableEventDistribution.
1.5 WebSphere adapters

In the following sections, we discuss the configuration options that are relevant to tuning J2EE™ Connector architecture (JCA) adapter applications.

1.5.1 Configure poll period and poll quantity

Two of the most important configuration parameters for the WebSphere Adapters are poll period and poll quantity. You can modify these settings in the activation specification of the adapter application. These settings only affect the performance of polling (inbound) applications.

- **Poll period** specifies the amount of time (in milliseconds) between polling actions.
- **Poll quantity** specifies the maximum number of events to process during a polling action.

Since these parameters control the rate and amount of work that an adapter processes, the combination of poll period and poll quantity regulate the number of transactions that are processed first by the adapter and then by the broker (WebSphere Process Server). As such, these parameters influence the performance of the entire solution, not just the adapter.

Non-optimal values for these parameters can result in low system throughput (if poll period is too long, poll quantity is too low, or both). Or it can cause excessive memory usage (and potentially OutOfMemory exceptions) if the parameters are configured to deliver events to the system at rates that are higher than the solution is implemented to handle (if poll period is too short, poll quantity is too high, or both).

Since both of these conditions dramatically impact overall system performance, appropriate settings for poll period and poll quantity are critical. They should be explicitly configured to support the level of throughput that a solution is designed to handle.

There are two possible scenarios to keep in mind. In the first scenario, the size of the objects is small (smaller heap memory requirements for objects in-flight), and throughput is of the essence. In this case, we can ensure that the adapter is not the bottleneck by reasonable over-polling. In the second scenario, the size of the business objects is large (bigger heap memory requirements for objects in-flight), and the criteria is to limit the number of objects in-flight at any moment in time to avoid OutOfMemory conditions. In general, for these scenarios, we recommend that you configure poll period and poll quantity to enable events to be retrieved and processed at a level that matches the peak throughput of the overall solution.

For example, if the peak throughput rate of a solution is 20 events per second, and events are continually available to the adapter for processing, set poll period to a small value, such as 10 milliseconds, and set poll quantity to 20. This supports the required peak throughput, while requiring a relatively small number of events to be concurrently held in the adapter process. The poll period enables a minimal delay between poll cycles.

Factors that may require an adjustment to these values include:

- **The size of the object being processed**
  
  For larger objects, a good rule of thumb is to use a lower poll quantity and longer poll period. This does not generally apply to relatively small objects (100 KB or less). However, for larger objects, it is important to minimize the number of objects held concurrently in the adapter process (in-flight), to avoid potential OutOfMemory exceptions.

  To extend the example, if the object size is 1 MB and the throughput rate of the solution is two events per second, appropriate settings can be poll period = 100 milliseconds and poll quantity = 2. Also note that the poll period can be more finely adjusted by calculating the
time it takes to process the poll quantity events (2 in this case) and adjusting the poll period for that.

- The Java heap size and physical memory that are available on the system

In general, the larger the heap is, the higher you can set the poll quantity. However, several factors are involved in setting the heap size. One important factor is to ensure that the heap is not set so large that paging results. Paging the Java heap dramatically reduces system performance. See 1.9, “Java” on page 19, for a detailed discussion about setting the Java heap sizes appropriately.

**Note:** The heap requirement for an object is really a multiple of its size. A good rule of the thumb is to use three times the size of the object, but it may vary depending on the adapter and the business process.

- The uniformity of event arrival rates

The previous examples assume that events arrive at a relatively constant rate. This may not be true for many solutions; event arrival is sometimes uneven. In these cases, use care to balance processing events in a timely manner in order to handle the occasional high arrival rates. At the same time, do not hold too many events in memory concurrently and potentially encounter OutOfMemory exceptions.

### 1.5.2 Configure the application server thread pool

For event delivery when using the default thread pool, increasing the number of threads that are available to the adapter can improve performance. The number of threads should be set to a value that includes the needs of the server and applications plus the JCA adapter.

1. Select **Servers** → **Application servers**.
2. Select your server, for example, server1.
3. In Additional Properties, select **Thread Pools**.
4. Select **Default** and update the Maximum and Minimum values.
5. Select **OK** and save the changes.

These settings, along with pollQuantity and pollPeriod, allow more objects to be in-flight at a single time (for polling adapters). Therefore, be careful not to set the value too large if your adapter processes large objects. Start with a setting of about 25 and tune to your configuration as required.

### 1.5.3 Configure the work manager thread pool

For adapters that perform request processing by using their own thread pool or the default work manager, the number of threads in the pool need to be tuned.

From the administrative console:

1. Select **Resources**.
2. Select **Asynchronous beans**.
3. Select **Work Managers**.
4. Select **DefaultWorkManager** (or your application-specific work manager).
5. Set the Minimum and Maximum number of threads.

We recommend that you specify 20 for both the minimum and maximum amounts as a starting point. Depending on your configuration, you might need more.
1.6 WebSphere Business Monitor

Figure 1-1 shows an example of a WebSphere Business Monitor installation. Some event emitters, such as business processes running on WebSphere Process Server, deliver events synchronously to CEI. CEI then delivers them to the JMS queue or queues that are associated with the monitor model or models that have registered to receive such events. Each business process’s events go to its dedicated queue. A Monitor Model (MM) corresponds to each business process. Monitor Server takes events from the queues, code that is specific to each model processes the events, and relevant information is added to the Monitor database.

The DB2 Replication Manager periodically replicates information from the Monitor database to the Datamart database and then deletes it from the Monitor database.

Monitor Dashboard offers various views of the data; the views are implemented as portlets in WebSphere Portal. Customers can direct queries to the Datamart database from a browser via Monitor Dashboards. Dashboard translates certain customer activities at the Dashboard into SQL or MDX queries directed at the Datamart database.

The default initial and maximum heap sizes in most implementations of Java are too small for many of the servers in this configuration. The Monitor Launchpad installs Monitor and its prerequisite servers with larger heap sizes. You should check whether these sizes are appropriate for your hardware and workload. We used initial and maximum heap sizes of 1024M and 1500M, respectively, on all our WebSphere servers during performance testing.

1.6.1 Configure CEI

By default, when an event arrives at CEI, it is placed into the queue that you specified (in this case, into a queue that the Monitor Launchpad specified) and into an additional, default queue. Unless the customer needs this extra queue, do not double-store.
You can prevent this extra store by following these steps in the WebSphere Application Server administrative console (using the Cell scope):

1. Select **Resources → Common Event Infrastructure Provider → Event Group Profile List → Event groups list → Event Group Profile**.
2. Select the **All Events** check box from the list of groups, and then click the **Delete** button.

The JMS queue provided by CEI, into which processes place BPEL events and out of which Monitor takes events, is persistent. By default, the persistence is provided by the messaging engine, which in turn uses the Cloudscape database management system. We have found that performance can be improved by replacing Cloudscape in this role with DB2.

Even though the CEI queue is persistent, CEI offers the ability to explicitly store events in a database. This is expensive, and unless the customer needs this extra copy, do not save it.

You can disable the CEI data store by following these steps in the WebSphere Application Server administrative console (using Cell scope):

1. Select **Resources → Common Event Infrastructure Provider → Event Server Profile → Default Common Event Infrastructure event server**.
2. Deselect **Enable Data Store**.

If you have CEI and Monitor Server on separate platforms, as shown in Figure 1-1 on page 14, and if they are connected over a foreign bus, then the following settings can significantly speed communication:

1. Add the following lines to the **ProcServerHome/properties/sib.properties** file of the CEI server:
   
   ```
   com.ibm.ws.sib.jfapchannel.maxPriorityQueueDepth=50000
   com.ibm.ws.sib.jfapchannel.maxPriorityQueueBytes=8388608
   ```

2. Add the names and values shown in Table 1-6 to the custom properties for both the CEI messaging engine used for sending Monitor events and the Monitor messaging engine on the Monitor Server.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>sib.msgstore.cachedDataBufferSize</td>
<td>8388608</td>
</tr>
<tr>
<td>sib.msgstore.discardableDataBufferSize</td>
<td>8388608</td>
</tr>
</tbody>
</table>

   You can add these properties using the WebSphere Application Server administrative console by selecting **Service integration → buses → Monitor bus name → Messaging engines → messaging_engine → Custom properties**.

### 1.6.2 Set the event processing batch size

Taking events off the CEI queue in large batches is much more efficient than doing it one at a time. Up to some limit, the larger the batch size is, the higher the event processing throughput is. But there is a trade-off: Taking some events off the CEI queue, processing them, and persisting them to the Monitor database is done as a transaction. While a larger batch size yields better throughput, it costs more if you have to roll back.

Monitor Launchpad sets the batch size to 100 events, which is the value we used during performance testing. If you experience frequent rollbacks, consider reducing the batch size. You can do this in the WebSphere Application Server administrative console (using the server Name Value

---

**Table 1-6 Custom properties for the CEI and Monitor messaging engines**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>sib.msgstore.cachedDataBufferSize</td>
<td>8388608</td>
</tr>
<tr>
<td>sib.msgstore.discardableDataBufferSize</td>
<td>8388608</td>
</tr>
</tbody>
</table>
scope). Select **Applications** → **Monitor Models** → **version** → **RuntimeConfiguration** → **General** → **Event Processing Batch Size**: <default 100>.

Monitor processes events most efficiently when the Deserialization WorkManager work request queue is the same size as the event processing queue. If you change one, you should change both.

In the WebSphere Application Server administrative console:

2. Set Work request queue size to the same as the batch size.
3. Ensure that **growable** is not selected.

### 1.7 Database: General

In this section, we discuss performance tuning tips for databases in general.

#### 1.7.1 Place database log files on a fast disk subsystem

Databases are designed for high availability, transactional processing, and recoverability. Since database transactions may not be committed immediately, updates to specific database tables are made using a set of database log files. Updates are made to database log files after every transaction is committed. As a result, database log files may be heavily used. Therefore, read and write access performance to the database log files is critical to overall system performance. We recommend that you place database log files on a fast disk subsystem with write back cache.

#### 1.7.2 Size database cross-referencing tables correctly

Relationship cross-reference tables for identity relationships and other dynamic relationships grow continuously and can become quite large in a production environment. Use care to size these tables accordingly. We recommend that the first extent of the table contains as many rows as possible, even the entire table. This avoids extent interleaving and enables faster database service times for relationship lookups.

Currently there is no mechanism to specify extent sizes and physical storage attributes for these tables through the administrative console. Therefore, if these tables are expected to be large, the recommended approach is to first export the data in these tables and then to drop the tables. After that, recreate the tables according to the desired storage parameters and, finally, reload the data.

Determining which tables in the WebSphere Process Server database are the cross reference tables is straightforward. Each relationship participant has a cross-reference table dedicated to itself. For example, if there is a relationship definition for a dynamic relationship named Customer with participants SAP®, Clarify, and PeopleSoft®, there are three distinct tables in the database, one for each participant.
1.7.3 Place logs on a separate device from table spaces

A basic strategy for all database storage configurations is to place the database logs on separate physical disk devices from the tablespace containers. This reduces disk access contention between I/O to the tablespace containers and I/O to the database logs, as well as improving recoverability.

1.8 Database: DB2-specific

Providing a comprehensive DB2 tuning guide is beyond the scope of this paper. However, there are general rules of thumb that can assist in improving the performance of DB2 environments. In the following sections, we discuss these rules and provide pointers to more detailed information.

You can find a quick reference for DB2 performance tuning on the Web at:


For the complete set of current DB2 manuals, as well as database tuning guides, refer to the DB2 Information Center at:

http://publib.boulder.ibm.com/infocenter/db2help/index.jsp

1.8.1 Maintain current indexes on tables

While the WebSphere Business Integration products create a set of database indexes that are appropriate for many installations, additional indexes may be required in some circumstances. A database environment that requires additional indexes often exhibits performance degradation over time; in some cases, the performance degradation can be profound. Environments that need additional indexes often exhibit heavy read I/O on devices that hold the tablespace containers.

To assist in determining which additional indexes can improve performance, DB2 provides Design Advisor. Design Advisor is available from the DB2 Control Center, or you can start it from a command line processor. It has the capability to help define and design indexes that are suitable for a particular workload.

1.8.2 Update catalog statistics

It is important to update the DB2 catalog statistics so that DB2 can optimize accesses to key tables. Catalog statistics are used by the DB2 query optimizer to determine the access plan for evaluating a query. Statistics are maintained on tables and indexes. Examples of statistics include the number of rows in a table and the number of distinct values in a certain column of a table.

DB2 8.2 contains functionality called the DB2 Automatic Table Maintenance feature, which runs the RUNSTATS command in the background as required to ensure that the correct statistics are collected and maintained. By default, this feature is not enabled. It may be turned on from the DB2 Control Center. Updating statistics allows the DB2 query optimizer to create better performing access plans for evaluating queries.
To manually update statistics on all tables in the database, use the REORGCHK command. Example 1-2 shows a sample AIX script to accomplish this task (on database DBNAME).

**Example 1-2  DB2 commands to manually update statistics on all tables**

```sql
db2 connect to DBNAME
db2 -v reorgchk update statistics on table all
db2 connect reset
db2rbind DBNAME -l /tmp/rbind.log all -u db2inst1 -p password
db2stop
db2start
```

This script first asks DB2 to generate the statistics and then rebinds the database, incorporating any changes in indexes. The database restart is not strictly required. However, you should run the REORGCHK command and rebind when the system is relatively idle so that a stable sample may be acquired and to avoid possible deadlocks in the catalog tables.

For more information about maintaining catalog statistics, see the *IBM DB2 Universal Database™ Administration Guide: Performance* on the Web at:


### 1.8.3 Set buffer pool sizes correctly

A buffer pool is an area of memory into which database pages are read, modified, and held during processing. Buffer pools improve database performance. If a needed page of data is already in the buffer pool, that page is accessed faster than if the page had to be read directly from disk. As a result, the size of the DB2 buffer pools is critical to performance.

The amount of memory used by a buffer pool depends upon two factors: the size of buffer pool pages and the number of pages allocated. Buffer pool page size is fixed at creation time and may be set to 4, 8, 16, or 32 KB.

The most commonly used buffer pool is IBMDEFAULTBP, which has a 4 KB page size. Starting with version 8 of DB2, we recommend that you explicitly set the number of pages used by a buffer pool. You can do this by using either the CREATE BUFFERPOOL or the ALTER BUFFERPOOL commands. The default number of pages is 250, resulting in a small total default buffer pool size of 4 KB x 250 = 1000 KB.

**Important:** All buffer pools reside in database global memory, which is allocated on the database machine. The buffer pools must coexist with other data structures and applications, all without exhausting available memory. In general, having larger buffer pools improves performance up to a point by reducing I/O activity. Beyond that point, allocating additional memory no longer improves performance.

To choose appropriate buffer pool size settings, monitor database container I/O activity by using system tools or by using DB2 buffer pool snapshots. Avoid configuring large buffer pool size settings, which lead to paging activity on the system.
1.9 Java

Because the WebSphere Business Integration product set is written in Java, the performance of the JVM has a significant impact on the performance delivered by these products. JVMs externalize multiple tuning parameters that may be used to improve both tooling and runtime performance. The most important of these are related to garbage collection, setting the Java heap size and configuring threading parameters. In this section, we discuss these topics in detail.

Note that the products covered in this paper use IBM JVMs on most platforms (AIX, Linux®, Windows, and so on) and the HotSpot JVMs on selected other systems, such as Solaris™. Vendor-specific JVM implementation details and settings are discussed as appropriate.

For more information, refer to the IBM Java 1.4.2 Diagnostics Guide on the Web at:


While there are more tuning parameters than those discussed in this section, most are for specific situations and are not of general use.

Refer to the following Web site if you want a useful summary of Java HotSpot™ VM options for Solaris:

http://java.sun.com/docs/hotspot/VMOptions.html

In addition, you can refer to the following Web site for a useful FAQ about the Solaris Java HotSpot VM:

http://java.sun.com/docs/hotspot/PerformanceFAQ.html#20

1.9.1 Set the heap and nursery size to handle garbage collection efficiently

*Garbage collection* (GC) is the process of freeing unused objects so that portions of the JVM heap may be reused. Because the Java language specification does not provide explicit delete() or free() byte codes, it is imperative to occasionally detect and delete objects that no longer have active references and free that space for reuse.

Garbage collection is triggered automatically when there is a request for memory, that is object creation, and the request cannot be readily satisfied from the free memory available in the heap (allocation failure). Garbage collection may also be activated programmatically via a Java class library `System.gc()` call. In this case, garbage collection occurs immediately and synchronously.

While the function provided by the HotSpot and IBM garbage collectors is the same, the underlying technology is different. For both JVMs, garbage collection takes place in three phases: mark, sweep, and an optional compact phase. The implementation of the garbage collection phases is different. This is mainly due to the fact that the HotSpot engine is what is known as a *generational collector*, while the IBM JVM (by default) is not.

For a detailed discussion of the HotSpot generational GC, see *Tuning Garbage Collection with the 1.3.1 Java Virtual Machine* on the Web at:

In its default configuration, the IBM JVM consumes its entire heap before a garbage collection is triggered.

With the generational collector of the HotSpot JVM, objects are initially allocated in a section of the heap called the *nursery*. If the object remains "live" (still referenced) over time, it is promoted to a different section of the heap called the *tenured area*. For the HotSpot JVM, a garbage collection is triggered when either the nursery or the full heap is consumed.

Whether a full heap GC or nursery GC is performed, the first phase is to mark all referenced objects in the region being collected. This leaves all unreferenced objects unmarked and the space they occupy free to be collected and reused. Following the mark phase, free chunks of memory are added to a free list. This phase is referred to as *sweeping*.

Occasionally following the sweep phase, a compact phase is performed. The compaction moves objects closer together to create larger contiguous free chunks. A number of triggers can cause compaction. For instance, after a sweep, if there is still not a large enough contiguous chunk of memory, then compaction executes. Also, for most `System.gc()` calls, a compaction is done. Relative to the other phases involved, compaction can be a time-consuming process and should be avoided if possible.

The IBM JVM has been optimized to avoid compactions, because this is an expensive process.

**Monitoring garbage collection**

To set the heap correctly, you must first determine how the heap is being used. This is done easily by collecting a *verbosegc* trace. A verbosegc trace prints garbage collection actions and statistics to stderr. The verbosegc trace is activated by using the Java runtime option of `verbose:gc`. Output from verbosegc is different for the HotSpot and IBM JVMs, as shown by the following examples.

Example 1-3 shows the IBM JVM verbosegc trace output.

```
Example 1-3  IBM JVM verbosegc trace output

<AF[8]: Allocation Failure. need 1572744 bytes 5875 ms since last AF>
<AF[8]: managing allocation failure, action=1 (23393256)/131070968)
(2096880/3145728)>
<GC: Tue Dec 18 17:32:26 2001
<GC(12): freed 75350432 bytes in 168 ms, 75% free (100840568)/134216696)>
<GC(12): mark: 129 ms, sweep: 39 ms, compact: 0 ms>
<GC(12): refs: soft 0 (age >= 32), weak 0, final 0 , phantom 0>
<AF[8]: completed in 203 ms>
```

Example 1-4 shows the Solaris HotSpot JVM verbosegc trace output (young and old).

```
Example 1-4  Solaris HotSpot JVM verbosegc trace output

[GC 325816K → 83372K(776768K), 0.2454258 secs]
[Full GC 267628K → 83769K <- live data (776768K), 1.8479984 secs]
```
Using the IBM JVM output shown Example 1-3, you can see the following items:

- The metric following the word need is the size of the failed allocation that caused the garbage collection. On the same line, the amount of time in milliseconds since the last allocation failure is given.
- The next line with the <AF[8]> tag displays the amount of free space in the heap and in an area of the heap referred to as the “wilderness”. In the example, the line reports 23393256 free bytes out of a possibly 131070968 bytes. The (2096880/3145728) refers to the wilderness area free, which is usually ignored.
- The next set of lines provides information about the garbage collection that was caused to satisfy the allocation failure. The first line is a time stamp.
- This is followed by a line that includes the time to complete the GC, 168 ms, and the amount of free space after the GC, 75%. Both of these metrics are extremely useful in understanding the efficiency of the garbage collection and the heap usage.
- Following this line is a line that describes the time for the different components of the GC. Make sure that the number that follows compact is normally 0. That is, a well-tuned heap avoids compactions.
- For the GC, there is a line about soft, weak, and phantom references, as well as a count of finalizers.
- This is then bracketed by a line with a time for the full allocation failure.

### Setting the heap size for most configurations

This section contains guidelines for determining the appropriate Java heap size for most configurations. If your configuration requires that more than one JVM runs concurrently on the same system (for example, if you run both WebSphere Process Server and WebSphere Integration Developer on the same system), then you should also read “Setting the heap size when running multiple JVMs on one system” on page 22.

For many applications, the default heap size setting for the IBM JVM is sufficient to achieve good performance. In general, the HotSpot JVM default heap and nursery size is too small and should be increased (we show how to set these parameters later). For optimal performance and for applications with unpredictable loads or large live sets (a large number of Java objects that are active concurrently), optimize the heap size.

There are several approaches to setting the optimal heap sizes. We describe here the approach that most applications should use when running the IBM JVM on AIX. The essential information can be applied to other systems.

First, we provide more background. The IBM JVMs have a feature that deals with dynamically growing the heap; it is referred to as rate-trigger heap growth. This process attempts to set the size of the heap so that pauses are not too long and GC does not take too much time. This is done dynamically, and it adjusts with the workload. If too much time is being used in GC, the heap grows. If the heap is mostly free, the heap can shrink.

In order to use rate-trigger heap growth effectively, set the initial heap size (-ms option) to something reasonable (for example, 256 MB) and the maximum heap size (-mx) option to something reasonable, but large (for example, 1024 MB). Of course, the maximum heap size should never force the heap to page. It is imperative that the heap always stays in physical memory. The JVM then tries to keep the GC time to something reasonable behind the covers by growing and shrinking the heap. The output from verbosegc should then be used to monitor the GC actions.
You can use a similar process to set the size of HotSpot heaps. In addition to setting the minimum and maximum heap size, you should increase the nursery size to approximately one-fourth of the heap size.

**Important:** You should never increase the nursery to more than half the full heap.

The nursery size is set using the MaxNewSize and NewSize parameters (that is, `XX:MaxNewSize=128m, XX:NewSize=128m`).

After the heap sizes are set, use the verbosegc traces to monitor the GC actions. If you find something unpleasant from the verbosegc trace, you can modify the heap settings accordingly.

For example, if the percentage of time in GC is high and the heap has grown to its maximum size, you may increase throughput by increasing the maximum heap size. As a rule of thumb, greater than 10% of the total time spent in GC is generally considered “high”.

**Attention:** Increasing the maximum size of the Java heap may not always solve this type of problem because it is can be a memory over-usage problem. If the pause time is too long, decrease the heap size. If both problems are observed, an analysis of the application heap usage is required.

### Setting the heap size when running multiple JVMs on one system

Each running Java program has a heap associated with it. Therefore, if you have a configuration where more than one Java program is running on a single physical system, setting the heap sizes appropriately is of particular importance.

An example of one such configuration is when many the WebSphere Integration Developer is on the same physical system as WebSphere Process Server. Each of these is a separate Java program that has its own Java heap. If the sum of all of the virtual memory usage (including both Java heaps as well as all other virtual memory allocations) exceeds the size of physical memory, the Java heaps are subject to paging. As previously noted, this causes total system performance to degrade significantly.

To minimize the possibility of this occurring, use the following guidelines:

- Collect a verbosegc trace for each running JVM.
- Based on the verbosegc trace output, set the initial heap size to a relatively low value. For example, assume that the verbosegc trace output shows that the heap size grows quickly to 256 MB and then grows more slowly to 400 MB. Based on this, set the initial heap size to 256 MB (ms256m).
- Based on the verbosegc trace output, set the maximum heap size appropriately. Use care to ensure this value is not set too low; otherwise, OutOfMemory errors will occur. The maximum heap size must be large enough to allow for peak throughput. Using the previous example, a maximum heap size of 768 MB might be appropriate (mx768m).
- Be careful to not set the heap sizes too low, or garbage collections will occur frequently, which might reduce throughput. Again, a verbosegc trace assists in determining this. A balance must be struck so that the heap sizes are large enough that garbage collections do not occur too often, while still ensuring that the heap sizes are not cumulatively so large as to cause the heap to page. This balancing act, of course, is configuration dependent.
Summary of setting heap sizes
The following list summarizes our recommendations about setting heap sizes:

- Make sure that the heap never pages. That is, on a given system, the sum of all the JVM maximum heap sizes must fit in physical memory.
- Collect and analyze a verbosegc trace in order to optimize memory usage.
- Aim for less than 10% execution time to be spent in garbage collection. Analyze the verbosegc trace in order to determine the GC execution time. Object reuse and heap size tuning can help in this area.
- For optimal performance, the heap should be run with less than 60%, possibly even 50%, occupancy. This is readily determined from the verbosegc trace output.
- Avoid finalizers. A developer can never be guaranteed when a finalizer will run, and often they lead to problems. If you use finalizers, try to avoid allocating objects in the finalizer code. An IBM JVM verbosegc trace shows if finalizers are being called.
- Avoid compaction where possible. A verbosegc trace shows if compaction is occurring.
- Analyze requests for large memory allocations and then devise a method for reusing the object.
- Increase the size of the nursery for the Sun HotSpot JVM. A good rule is to set the nursery to one-fourth the size of the heap.

1.9.2 Set AIX threading parameters

The IBM JVM threading and synchronization components are based upon the AIX POSIX compliant Pthread implementation. The environment variables in Example 1-5 have been found to improve Java performance in many situations and have been used for the workloads in this document. The variables control the mapping of Java threads to AIX native threads, turn off mapping information, and allow for spinning on mutex locks.

Example 1-5   Setting AIX threading parameters

```bash
export AIXTHREAD_COND_DEBUG=OFF
export AIXTHREAD_MUTEX_DEBUG=OFF
export AIXTHREAD_RWLOCK_DEBUG=OFF
export AIXTHREAD_SCOPE=S
export SPINLOOPTIME=2000
```

You can learn more information about AIX specific Java tuning on the Web at:
- Running your Java application on AIX, Part 1: Getting started
- Running your Java application on AIX, Part 2: JVM memory models

1.9.3 Reduce or increase heap size if OutOfMemory errors occur

The java.lang.OutOfMemory exception is used by the JVM in a variety of circumstances, making it sometimes difficult to track down the source of the exception. There is no conclusive mechanism for telling the difference between these potential error sources, but a good start is to collect a trace using verbosegc. If the problem is a lack of memory in the heap, then this is easily seen in this output. See 1.9.1, “Set the heap and nursery size to handle garbage collection efficiently” on page 19, for further information about verbosegc output. Many
garbage collections resulting in little free heap space occur before the exception. If this is the problem, then increase the size of the heap.

However, if there is enough free memory when the java.lang.OutOfMemory exception is thrown, the next item to check is the finalized count from the verbosegc (only the IBM JVM gives this information). If the number appears high, then a subtle effect may be occurring whereby resources outside the heap are held by objects within the heap and are being cleaned by finalizers. Reducing the size of the heap can alleviate this situation, by increasing the frequency with which finalizers are run.

OutOfMemory errors can also occur for issues unrelated to JVM heap usage, such as running out of certain system resources. Examples of this include insufficient file handles or thread stack sizes that are too small. In some cases, you can tune the configuration to avoid running out of native heap:

- Try reducing the stack size for threads (the -Xss parameter). However, deeply nested methods may force a thread stack overflow in case of insufficient stack size.
- For middleware products, if you are using an in-process version of the driver, it is usually possible to find an out-of-process driver that can have a significant effect on the native memory requirements. For example, you can use Type 4 JDBC drivers (DB2 “Net” drivers, Oracle® “Thin” drivers), MQSeries® can be switched from bindings mode to client mode, and so on. Refer to the documentation for the products in question for more details.
Chapter 2. Best practices

This chapter provides guidance on how to develop and deploy WebSphere Process Server and WebSphere Enterprise Service Bus (WebSphere ESB) solutions for high performance. This guidance is based on the current implementation of the WebSphere Process Server and WebSphere ESB runtimes and WebSphere Integration Developer tooling. It will likely evolve as these products evolve and mature. While many of these best practices were developed as a result of the performance testing in the lab, we recognize that they are not always applicable or feasible. You should consider these practices as guidelines when developing new solutions.

The purpose of this chapter is to highlight the best practices that are associated specifically with the technologies and features that are delivered in the products that are covered in this report. However, these products are built on top of existing technologies such as WebSphere Application Server, Service Integration technology, and the Business Process Choreographer. Each of these technologies has associated best practices that still apply. It is not our intent to enumerate these here. Instead, see “Related publications” on page 41 for a set of references and pointers to this information.
2.1 Modeling

In this section, we discuss design issues to consider during application development.

2.1.1 Cache results of ServiceManager.locateService()

When writing Java code to locate a Service Component Architecture (SCA) service, either within a Java component or a Java snippet, consider caching the result for future use, because service location is a relatively expensive operation.

2.1.2 Avoid overuse of WebSphere Process Server components

WebSphere Process Server components are services. As such, they are inherently more expensive to locate and invoke than method invocations, for example. When developing solutions, aim for the coarsest granularity to minimize the service invocation overhead.

2.1.3 Use the simplest component kind for the required function

WebSphere Process Server provides valuable functional building blocks that reduce the complexity of creating business logic. Such features as business rules, selectors, and business state machines are examples of this value. However, as in any programming system, value comes at a cost, and the developer should be mindful of the performance trade-offs.

All else being equal, a simple Java condition in a Java component or a Java snippet is more efficient than invoking a business rule. However, business rules provide a declarative and expandable way to specify business decisions, the benefits of which may override the performance overhead. Such trade-offs are a natural part of any solution development and should be made here as well.

2.1.4 Reduce the number of SCA modules when appropriate

WebSphere Process Server components can be assembled into modules for deployment. When assembling modules we recognize that many factors come into play. While more modules can allow for better distribution across servers and nodes, modularization also has a cost. When components are placed together in a single server instance, it is best to package them within a single module for best performance.

2.1.5 Prefer synchronous SCA bindings across local modules

For cross-module invocations, where the modules are likely to be deployed locally, for example, within the same server JVM, we recommend that you use the synchronous SCA binding. This binding has been optimized for module locality and outperforms other bindings.

**Note:** Synchronous SCA is as expensive as other bindings when invocations are made between modules located in different WebSphere Process Server or WebSphere ESB servers.
2.1.6 Use multi-threaded SCA clients to achieve concurrency

Synchronous components that are invoked locally, for example, from a caller in the same server JVM, execute on the context of the caller's thread. Thus, if desired, concurrency must be provided by the caller in the form of multiple threads.

2.1.7 Prefer non-interruptible processes over interruptible processes

If call targets respond quickly, use non-interruptible processes if possible in conjunction with synchronous invocation.

If call targets respond slowly or the process contains human tasks, use interruptible processes. Consider implementing a process hierarchy with non-interruptible processes for heavily used parts of the overall business scenario and interruptible processes for human tasks, longer running asynchronous tasks, or both.

2.1.8 Choose the appropriate granularity for a process

A business process and its individual steps should have “business significance” and not try to mimic programming level granularity. Use programming techniques such as Plain Old Java Objects (POJOs), Enterprise JavaBeans™ (EJB™), and servlets for logic without business significance.

2.1.9 Join activities into one transaction if possible

Long-running business processes can run in multiple transactions. The border of those transactions and the number of transactions can be changed with transactional behavior settings on Java snippet, Staff, and Invoke activities. By default, each activity runs in its own transaction. For many types of activities, this is unnecessary.

For example, a preceding assign can safely run in the transaction of a succeeding invoke. This is achieved by setting the Transactional behavior property of the assign activity to participate. The default setting for this property is commit after, which causes each activity to run in its own transaction.

Adjust the transactional behavior settings such that the resulting execution of the long-running business process performs best. As a rule of thumb, fewer transactions result in better performance.

2.1.10 Minimize variable specifications

Use as few variables as possible and minimize the size of the business objects used. In long-running processes, each commit saves modified variables to the database (to save context), and multiple variables or large business objects make this costly.

2.1.11 Increase the maximum batch size on activation specifications

The maximum batch size can be set in the activation specification. The default value is 1. The maximum batch size value determines the number of messages that are taken from the messaging layer and delivered to the application layer in a single step.
Increasing the maximum batch size value, for example to 8, for activation specs associated with long-running business processes can improve performance.

2.1.12 Set the preferred interaction style to Sync whenever possible

Many WebSphere Process Server component types, such as interface maps, invoke their target components based on the target's setting of preferred interaction style. Since synchronous cross-component invocations are better performing, we recommend that you set preferred interaction style to Sync whenever possible. Only in specific cases, for example when invoking a long-running business process, or more generally whenever the target component requires asynchronous invocation, should this be set to Async.

2.1.13 Use asynchrony judiciously

WebSphere Process Server components and modules may be wired to each other either synchronously or asynchronously. The choice of interaction style can have a profound impact on performance. Use care when making this choice.

2.1.14 Avoid unnecessary cross-component asynchronous invocations within a module

It is important to realize that asynchronous invocations in WebSphere Process Server are intended to provide a rich set of qualities of service, including transactions, persistence, and recoverability. Therefore, an asynchronous invocation should be thought of as a full messaging hop to its target. When the intended target of the invocation is in the same module, a synchronous invocation yields much higher performance.

2.1.15 Set MaxSession for the MQ/JMS export binding

When an MQ/JMS export binding is created, the maximum number of sessions property in Binding → End-point configuration → listener ports properties is initially set to 1 in WebSphere Integration Developer. This MaxSession property informs the container how many concurrent incoming MQ/JMS requests can be processed at the same time. For better concurrency, you would change this property to a larger value, for example 10.

2.1.16 Set MQ MaxSession for the MQ export binding

The implementation of the MQ export binding relies on a message-driven bean (MDB) over MQ/JMS. However, the listener port properties are not visible in the WebSphere Integration Developer. The corresponding MaxSession property of the listener port of an MQ export binding needs to be changed after the application is installed. From the WebSphere administrative console, locate the corresponding listener port, and change the MaxSession property to a larger value if needed. The current default setting (in WebSphere Process Server and WebSphere ESB 6.0.2) is 5.

Note: While this work happens in a single step, it does not mean that it is done within a single transaction. Therefore, this setting does not influence transactional scope.
2.1.17 Choose efficient metadata management

In this section, we discuss techniques for using metadata efficiently.

Follow Java language specification for complex data type names
While WebSphere Process Server allows characters in business object type names that are not permissible in Java class names (the '_' for example), the internal data representation of complex data type names does occasionally use Java types. As such, performance is better if business object types follow the Java naming standards.

Avoid use of anonymous derived types in XSDs
Some XSD features (restrictions on the primitive string type, for example) result in modifications to the type that require a new sub-type to be generated. If these types are not explicitly declared, then a new sub-type (a derived type) is generated at run time. Performance is generally better if you can avoid this.

Therefore, avoid adding restrictions to elements of primitive type where possible. If a restriction is unavoidable, consider creating a new, concrete SimpleType that extends the primitive type to include the restriction. Then XSD elements may use that type without degraded performance.

Avoid referencing elements from one XSD in another XSD
For example, A.xsd defines an element called AElement as shown in Example 2-1.

Example 2-1 AElement definition referenced from another file

```xml
<xs:element name="AElement">
  <xs:simpleType name="AElementType">
    <xs:restriction base="xs:string">
      <xs:minLength value="0" />
      <xs:maxLength value="8" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

AElement may then be referenced from another file, B.xsd, as:

```xml
<xs:element name ref="AElement" minOccurs="0" />
```

This type of reference has been shown to perform poorly. It is much better to define the type concretely and then make any new elements use this type. Therefore A.xsd is defined as shown in Example 2-2.

Example 2-2 AElement concrete definition

```xml
<xs:simpleType name="AElementType">
  <xs:restriction base="xs:string">
    <xs:minLength value="0" />
    <xs:maxLength value="8" />
  </xs:restriction>
</xs:simpleType>
```

Then B.xsd becomes:

```xml
<xs:element name="BElement" type="AElementType" minOccurs="0" />
```
Reuse data object type metadata where possible
Within application code, it is common to refer to types, for instance when creating a new business object. It is possible to refer to a business object type by name, for example, in the method `DataFactory.create(String uri, String typeName)`. It is also possible to refer to the type by a direct reference as in the method `DataFactory.create(Type type)`. In cases where a Type is likely to be used more than once, it is usually faster to retain the Type (for instance, via `DataObject.getType()`) and reuse that type for the second and future uses.

2.1.18 Recommendations modified since the 6.0.1 version of the report
Beginning with version 6.0.1.1 of WebSphere Process Server, the overhead associated with the Java Message Service (JMS) and asynchronous SCA invocations has been greatly reduced. Because of these enhancements, we have revised the following recommendations.

Cross-module asynchronous invocations
Prior to WebSphere Process Server version 6.0.1.1, the cost of a cross-module asynchronous invocation was greater than that of an intra-module asynchronous invocation followed by a cross module synchronous invocation.

Due to internal efficiency improvements in WebSphere Process Server 6.0.1.1, the situation is now reversed. Better performance results from a cross-module asynchronous invocation.

Preferred interaction style
A previous recommendation was to set the preferred interaction style (sync or async) of any module called by an interface map component. Beginning with WebSphere Process Server version 6.0.1.1, interface map components no longer reflect the interaction style (sync or async) of the caller when invoking their target. Instead, they inspect the called preferred interaction style of the component.

2.1.19 Use asynchronous or 1-way interaction styles for macroflows
When designing long-running process components, ensure that the interface does not use 2-way (request/response) synchronous semantics, because this ties up the caller’s resources (thread, transaction, and so on) until the process completes. For long-running processes, this is highly undesirable. Instead, invoke such processes asynchronously or via “1-way” synchronous calls, where no response is expected.

2.2 Deployment
In this section, we discuss performance tuning tips for deployment.

2.2.1 Place local modules in the same server JVM
If you are planning to deploy WebSphere Process Server modules on the same physical server, you can achieve better performance by deploying the modules to the same application server JVM. Doing so allows the server to exploit this locality.
2.2.2 Configure MDB concurrency for asynchronous components

Asynchronous invocations of an SCA component use an MDB to listen for incoming events that arrive in the associated input queue. Threads for these MDBs come from the default thread pool for an application server.

Concurrency of all MDBs together is determined by the number of threads in the WebSphere Process Server default thread pool. Each application server has a default thread pool setting that can be set with the WebSphere administrative console under the Thread Pools category. Additionally, WebSphere Process Server can be configured so that MDB threads are taken from an aliased thread pool that can be defined through the WebSphere Process Server administrative console.

The number of concurrent instances of a particular MDB can be configured using the Maximum Concurrent Endpoints parameter. You can also set this parameter to use the WebSphere Process Server administrative console in the configuration section for individual MDBs.

One symptom of insufficient concurrency is CPU idleness. Vary the concurrency to achieve maximum CPU utilization and throughput.

2.2.3 Set the WorkArea service maximum send and receive sizes to no limit

Making synchronous SCA calls across JVMs results in the underlying infrastructure serializing the work area in the request and deserializes the work area from the reply. The default work area service as well as the WorkArea partition service uses the user-specified size limits as an upper limit for sizes that can be sent or received by a work area.

The possible values are 0 (no limit), -1 (default), or a non-zero positive number. It has been found that setting the sizes to 0 (no limit) is beneficial to performance because it circumvents the costly checking of the outbound or incoming work area size.

2.2.4 Use events judiciously

The purpose of the Common Event Infrastructure (CEI) event emission in WebSphere Process Server is for business activity monitoring. Since WebSphere Process Server CEI emission is a persistent mechanism, it is inherently heavy weight. You should enable CEI only for events that have a business relevance. What we have seen in practice appears to be confusion between business and IT monitoring, for which Performance Monitoring Infrastructure (PMI) is far more appropriate.

With this in mind, the following points generally hold for most customers:

- Businesses are concerned about the state of their business and their processes. Therefore events that signify changes in state are important. For long-running and human task activities, this is fairly natural. We want to detect when long-running activities complete, when human tasks change state, and so on.

- For short running flows that complete within seconds, it is usually sufficient to know that a flow completed, with perhaps the associated data. It usually makes no sense to distinguish events within a microflow that are only milliseconds or seconds apart. Therefore, two events (start, end) are usually sufficient for a microflow. In actuality, one event should suffice, but two are required currently in our implementation.
2.3 WebSphere Adapters

In this section, we discuss best practices when using WebSphere Adapters.

2.3.1 Monitor JMS queues and external event table

Certain adapters, such as FlatFile, FTP, e-mail, and SAP, can use external event tables. JMS queues may also be used at various points, for example during asynchronous event delivery. You should periodically monitor both the tables and JMS queues to ensure that events are being processed and removed from these queues. Obsolete events can produce delays in processing, since the adapter may attempt to unnecessarily recover these events.

You can find external table names by using the WebSphere Application Server administrative console. The names are specified in the J2C activation specification of the deployed adapter.

These tables can then be monitored and cleared with the appropriate database administration tool. For example, if DB2 is your database, then use the DB2 Control Center. You can also find the relevant JMS queues by using the WebSphere Application Server administrative console. Again, if you are using DB2, they are specified as data sources in the Service Integration and Resources sections.

The relevant tables and queues can be monitored while the adapter is running. If you detect that events need to be removed from the tables or JMS queues, stop the application that is using the adapter first. Then, remove the events, and restart the application.

2.3.2 Configuring synchronous event delivery

By default, event delivery is done by using asynchronous semantics. Synchronous semantics are also available for event delivery and typically provide significantly greater throughput. Whenever it is reasonable to use either synchronous or asynchronous semantics, we recommend that you use synchronous for better performance. You do this in WebSphere Integration Developer tooling and cannot change it at deployment time.

To set event delivery to use synchronous semantics in WebSphere Integration Developer 6.0.2:

1. Open your application in the WebSphere Integration Developer and go to the Business Integration perspective.
2. Open the appropriate module using the Assembly Editor.
3. Right-click the adapter export box and then select Show in Properties. You see the Properties tab for the adapter export.
4. Select the Binding tab.
5. Select the Performance attributes tab.
6. Change the pull-down value in the Interaction Style box to sync.
7. Save your changes, rebuild your application, export your EAR file, and redeploy.
2.4 Large objects

An issue that field personnel encounter frequently is trying to identify the largest object size that both WebSphere Process Server and the corresponding adapters can effectively and efficiently process. Since there is not a simple answer to this question, we present both a discussion of the issues that are involved and some practical guidelines for the current releases of WebSphere Process Server and adapters.

In general, objects that are 5 MB or larger may be considered “large” and require special attention. For reasons detailed as follows, we recommend that, if these large objects must be supported, you architect the solution design to break the object into multiple objects sized at or below 1 MB and process them individually. This approach generally produces a more robust and a higher performing solution.

2.4.1 Factors that affect large object size processing

Stated at a high level, the object size capacity for a given installation depends on the size of the Java heap and the load placed on that heap (that is, the live set) by the current level of incoming work. The larger the heap is, the larger the business object is that can be successfully processed.

To apply this somewhat general statement, you must first understand that the business object size limit is based on the following fundamental implementation facts of JVMs:

- **Java heap size limitations**
  
  The limit to the size of the Java heap is operating system dependent. For further details of heap sizes, see “Heap limitations: Increase the Java heap to its maximum” on page 34. However, it is typical to have a heap size limit of around 1.4 GB.

- **Java heap fragmentation**
  
  In a production WebSphere Process Server environment, the server and adapters may have been operating for a long period before a large object arrives. All JVMs implement a heap compaction function that will limit, but cannot totally eliminate, heap fragmentation. Some objects in the Java heap cannot be moved.
  
  As a result, fragmentation always exists in the heap, potentially making the amount of contiguous memory available in a single block much less than the overall total of available memory. Java objects must be allocated in a contiguous space. This fragmentation is normally not an issue because most Java object requests are small and easily allocated, even in a fragmented heap.

- **Contiguity of Java objects**
  
  If a large Java object is requested, and after compaction and growing the heap to its maximum size, there is not a sufficiently large contiguous chunk of Java Heap available, then Java throws an OutOfMemory exception. This is a typical scenario encountered with very large business objects and brings to light two important facts:
    - First, even though there may be plenty of free space available in the heap, an OutOfMemory exception can still occur. We have seen instances in the field where half of a 1.6 GB heap was free (800 MB), but a 38 MB allocation caused an OutOfMemory exception. As previously described, this is due to the fact that the Java heap becomes fragmented over time.
    - Second, there is an issue concerning the relationship between WebSphere Process Server business objects and the Java objects that represent them. A business object that appears as a 10 MB message on a JMS input queue for an adapter, for example,
may result in the allocation of many larger Java objects as it flows through the WebSphere Process Server and adapters. That is, a business object that consumes 10 MB on the input message queue may result in allocations of 20 to 30 MB on the Java heap, since single-byte binary objects are generally converted to multi-byte character representations (for example, Unicode). The fact that a business object consumes more of the Java heap than the corresponding message may be due to growth in the mapping of objects as well as to the internal implementation details in the WebSphere Process Server and WebSphere Adapters.

### 2.4.2 Best practices to mitigate large object issues

It has been observed that the use of large objects in an architected solution is frequently associated with batch oriented business processes, where a large number of smaller transactions are assembled and submitted as a single object for processing. Given that, we must address the issues outlined in 2.4.1, “Factors that affect large object size processing” on page 33.

**Heap limitations: Increase the Java heap to its maximum**

We address the issue of Java heap size limitations by platform:

- **Windows**
  
  As mentioned previously, WebSphere Process Server 6.0.2 and WebSphere Adapters 6.0.2 ship with 1.4.2 JVMs from IBM on Windows platforms. Due to address space limitations in the Windows operating system, the largest heap that can be obtained is around 1.4 GB to 1.6 GB.

- **AIX**
  
  The current version of WebSphere Process Server 6.0.2 and WebSphere Adapters 6.0.2 support the 32-bit version of 1.4.2 JVMs from IBM on AIX. Using the normal Java heap settings, the 1.4.2 JVM supports heaps in excess of 3 GB. Refer to the article “Getting more memory in AIX for your Java applications” for a description of the interaction between the IBM JVM and the AIX virtual memory manager: [http://www-128.ibm.com/developerworks/eserver/articles/aix4java1.html](http://www-128.ibm.com/developerworks/eserver/articles/aix4java1.html)

- **Solaris**
  
  On Solaris, WebSphere Process Server 6.0.2 and WebSphere Adapters 6.0.2 ship with the 32-bit version of the 1.4.2 JVM from Sun Microsystems. Laboratory experiments indicate a maximum of approximately a 3.5 GB heap for WebSphere Process Server 6.0.2 workloads.

For IBM JVMs, the following settings enhance the ability to allocate large objects.

- **Set** `-Xms` **to half of** `-Xmx`, for example: `-Xms700m -Xmx1400m`
  
  These parameters are the initial and maximum size of the Java heap, respectively.

- **Set** `-Xminf` **and** `-Xmaxf` **to values such as**: `-Xminf0.2 -Xmaxf0.4`
  
  `-Xminf` **and** `-Xmaxf` control how the JVM expands or contracts the heap. The default settings of these parameters are set to maximize throughput performance by aggressively expanding the heap. Less aggressive heap expansion and more aggressive heap contraction helps to preserve the unused heap space for future expansion, thus enabling JVM to allocate large objects.

- **Set** `-Xloratio`, for example `-Xloratio0.4`
  
  `-Xloratio` attempts to preserve the specified fraction of the Java heap for large object allocations.
Reduce Java heap fragmentation: Limit concurrent processing

One way to significantly reduce the fragmentation in the Java heap and allow for larger object sizes is to limit the concurrent processing activity occurring within the JVM. Obviously, you should not expect to be able to process a steady stream of the largest objects possible concurrently with other WebSphere Process Server and WebSphere Adapters activities. The operational assumption that needs to be made when considering large objects is that not all objects will be “large” and that large objects do not arrive often, perhaps once or twice per day. If more than one large object is being processed by WebSphere Process Server concurrently, the likelihood of failure increases dramatically.

The size and number of the “normally arriving” smaller objects affects the amount of Java heap memory consumption, and possibly fragmentation as well, that exists within the WebSphere Process Server and adapters. Generally speaking, the heavier the load is on a system when a large object is being processed, the more likely memory problems will be encountered.

For adapters, the amount of concurrent processing can be influenced by setting the pollPeriod and pollQuantity parameters. To allow for larger object sizes, set a relatively high value for pollPeriod (for example, 1 second) and low value for pollQuantity (for example, 1) to minimize the amount of concurrent processing that occurs. Note that these settings are not optimal for peak throughput, so if a given adapter instance must support both high throughput for smaller objects interspersed with occasional large objects, then trade-offs must be made. For a detailed discussion on setting these parameters, see 2.3, “WebSphere Adapters” on page 32.

Additionally, for the mySAP™.com adapter when using the ALE module for event delivery, set the connector-specific NumberOfListeners configuration property to 1. This limits the number of IDOCs processed concurrently by the adapter.

Contiguity of Java objects: Send large objects as multiple small objects

If a large object needs to be processed on current WebSphere Process Server and adapter technologies, then the solutions engineer must find a way to limit the large Java objects being allocated. The primary technique for doing so involves decomposing large business objects into smaller objects and submitting them individually.

If the large objects are a collection of small objects as assumed previously, the solution is to group the smaller objects into conglomerate objects less than 1 MB in size. This has been done at a variety of customer sites for predecessor products. If there are temporal dependencies or an “all-or-nothing” requirement for the individual objects, then the solution becomes more complex. Implementations at customer sites using predecessor products have shown that dealing with this complexity is worth the effort as demonstrated by both increased performance and stability.

Certain adapters, such as the FlatFile JCA Adapter, can be configured to use a SplitBySize mode with a SplitCriteria set to the size of each chunk. In this case, a large object is split in chunks of the size that is specified by SplitCriteria to reduce peak memory usage.
2.5 WebSphere Business Monitor

In this section, we discuss best practices for WebSphere Business Monitor.

2.5.1 Event processing

If a Monitor Model application is turned off while events are being posted to its queue, the number of events in the queue will grow. The throughput performance of this particular implementation of a JMS queue degrades as the number of events in the queue grows. Therefore, a Monitor Model should always be running if events are being posted to its queue.

A significant portion of event processing is single-threaded. Thus each instance of Monitor Server can usefully employ no more than two CPUs. If performance on a given Monitor Model is not satisfactory on a given platform, we recommend that you move to a platform with faster processors, rather than moving to a platform with more processors. Multiple Monitor Models can each use two CPUs on a symmetric multiprocessor (SMP).

2.5.2 Dashboard

The platform requirements of the Dashboard, Alphablox, and Portal stack are relatively modest compared to those of Monitor Server and the database server. The most important consideration for good Dashboard performance is to size and configure the DB server correctly. Be sure it has enough CPU capacity for anticipated data mining queries, enough RAM for bufferpools and so on, and plenty of disk arms.

In all of our measurement configurations, we had Monitor, Datamart, and each of their logs, on separate RAID arrays.

The data sources in both Monitor Dashboard and WebSphere Portal use pools of connections to communicate with DB2. It is important that these pools be large enough that threads do not have to wait for a connection. There are two data sources in Dashboard: Monitor_state_DB and DatamartDBDataSource.

By default, WebSphere sets connection pool sizes to 10, but Monitor Launchpad initializes these two pools to 100, which we have found sufficient for the workloads described in this paper. However, if, under heavy Dashboard load, there are periods when nothing appears to be happening, try increasing these pool sizes.

For example, to set the Maximum connection pool size for DatamartDBDataSource in the WebSphere Application Server administrative console in Cell scope, select Resources → JDBC providers → MonitorDB2XADriver → Data sources → [Monitor_state_DB or DatamartDBDataSource] → Connection pools: Maximum connections: <default 100>.

2.5.3 Database server

Both event processing and Dashboard rely on a fast, well-tuned database server for good performance. The design of Monitor assumes that any customer who is using it has strong on-site DB administrator skills. We strongly recommend that you read and follow the database tuning advice and recommendations in 1.7, “Database: General” on page 16, and 1.8, “Database: DB2-specific” on page 17.
Measurement configurations

In this chapter, we provide suggested initial settings for several relevant parameters. These values will not be optimal in all cases, but we have found that these values work well in our internal performance evaluations. They are, at a minimum, useful starting points for many customer deployments.
3.1 WebSphere ESB settings

In this section, we discuss the settings that are used for internal performance evaluations of WebSphere Enterprise Service Bus (WebSphere ESB). You can consider these settings as useful starting points when using WebSphere ESB. Otherwise, the default settings supplied by the product installer can be used as a starting point.

3.1.1 WebSphere ESB common settings

These settings are used for all WebSphere ESB performance evaluations, for both Web services and Java Message Service (JMS).
- Tracing is disabled.
- Security is disabled.
- Java heap size min and max values are fixed at 1280 MB for Windows and 1024 MB for AIX.

3.1.2 WebSphere ESB settings for Web services tests

The following settings were used for the Web services performance evaluations:
- PMI monitoring is disabled.
- WebContainer thread pool sizes are set to a maximum value of 15 and minimum value of 10.
- WebContainer thread pool inactivity timeouts for thread pools is set to 3500.
- HTTP settings keepAliveEnabled is set to true, and maxPersistentRequests is set to -1.
- EJB cache properties for ejbCacheSize is set to 2053.
- Java2 Security is set to false.

3.1.3 WebSphere ESB settings for JMS tests

These settings were used for the JMS performance evaluations:
- For Activation specification, Maximum concurrent endpoints is set to 30.
- For Queue connection factory, the Maximum connection pool size is set to 30.
- DiscardableDataBufferSize is set to 10 MB, and CachedDataBufferSize is set to 40 MB.
- For MQ JMS Non Persistent Listener Port configuration:
  - Max Sessions is set to 10.
  - Max Messages is set to 10.
  - For Queue connection factory, the Maximum connection pool size is set to 30.
- For MQ JMS Persistent Listener Port configuration:
  - Max Sessions is set to 50.
  - Max Messages is set to 1.
  - For Queue connection factory:
    - The Maximum connection pool size is set to 51.
    - The Max session pool size to 50.
3.1.4 DB2 settings for WebSphere ESB JMS persistent tests

These settings are only relevant to the JMS persistent performance evaluations because they use the database to persist messages.

- Place database table spaces and logs on a fast disk subsystem.
- Place logs on a separate device from table spaces.
- Set the buffer pool size correctly.
- Set the connection min and max to 30.
- Set the statement cache size to 40.
- Use raw partition for DB2 logs.

Otherwise, unless specifically noted in the workload description, the default settings as supplied by the product installer were used.

3.2 WebSphere Business Monitor settings

The settings in Example 3-1 are DB2 8.2 settings used for internal performance evaluations of WebSphere Business Monitor. Consider these settings as a useful starting point when using WebSphere Business Monitor. Otherwise, the default settings supplied by the product installer can be used as a starting point.

Example 3-1  Monitor database settings

db2set DB2_EXTENDED_OPTIMIZATION=ON
db2set DB2_HASH_JOIN=y
db2set DB2_RR_TO_RS=YES
db2set DB2COMM=tcpip
db2set DB2AUTOSTART=YES
db2 update db cfg for Monitor using locklist 2048
db2 update db cfg for Datamart using locklist 4096
db2 update db cfg for Monitor using maxlocks 25
db2 update db cfg for Datamart using maxlocks 75
db2 update db cfg for Monitor using sortheap 4096
db2 update db cfg for Datamart using sortheap 8192
db2 update db cfg for Monitor using logprimary 10 logsecond 80 logfilsiz 5000
db2 update db cfg for Datamart using logprimary 10 logsecond 80 logfilsiz 5000
db2 update db cfg for Datamart using dft_degree 2
db2 update dbm cfg for Datamart using sheapthres 20000
db2 connect to Monitor
db2 alter bufferpool DSDFLTBP4 size 25000
db2 alter bufferpool DSDFLTBP8 size 25000
db2 connect reset
db2 connect to Datamart
db2 alter bufferpool DSDFLTBP4 size 150000
db2 alter bufferpool DSDFLTBP8 size 5000
db2 connect reset
3.3 Java settings

In this section, we list the Java settings that we used for internal performance evaluations. Consider these settings as a useful starting point. Otherwise, the default settings supplied by the product installer can be used as a starting point.

- General
  -`-verbosegc` (for gathering verbose garbage collection statistics)

- WebSphere Process Server
  -`-Xms512M -Xmx512M`

- WebSphere Business Monitor
  -`-Xms1024M -Xmx1500M`
Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this paper.

Online resources

- IBM WebSphere Business Process Management Version 6.0 information center
  http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/index.jsp
- Setting up the data store for a messaging engine
- Best practices for tuning DB2 UDB v8.1 and its databases
- A Quick Reference for Tuning DB2 Universal Database EEE
- DB2 Information Center
  http://publib.boulder.ibm.com/infocenter/db2luw/v8/index.jsp
- DB2 9 for Linux UNIX® and Windows
  http://www-306.ibm.com/software/data/db2/support/db2_9
- IBM DB2 UDB Administration Guide: Performance V8.2, SC09-4821
- WebSphere Application Server: Performance information
  http://www.ibm.com/software/webservers/appserv/was/performance.html
- Tuning Performance section of the WebSphere Application Server Information Center
  (Includes tuning for messaging)
- Extending a J2CA adapter for use with WebSphere Process Server and WebSphere Enterprise Service Bus
IBM WebSphere Developer Technical Journal: A guided tour of WebSphere Integration Developer -- Part 1

Securing adapters (WebSphere Enterprise Service Bus information center)

IBM WebSphere Developer Technical Journal: Web Services Architectures and Best Practices

Best practices for Web services: Part 1, Back to the basics

Best Practices for Web services: Part 9


WebSphere Journal: “The Dynamic Caching Services”
http://websphere.sys-con.com/read/43309.htm

IBM Java Diagnostics Guide 1.4.2

How to get IBM Redbooks publications

You can search for, view, or download IBM Redbooks, Redpapers, Technotes, and draft publications, and Additional materials, as well as order hardcopy IBM Redbooks publications, at this Web site:
ibm.com/redbooks

Help from IBM

IBM Support and downloads
ibm.com/support

IBM Global Services
ibm.com/services
Index

Numerics
1-way interaction style  30

A
activation specification  8, 27, 38
adapter event delivery tables  3
AIX threading parameters  23
Alphablox  36
ALTER BUFFERPOOL  18
anonymous derived types  29
Application Response Measurement (ARM)  3
application server thread pool  13
asynchronous event delivery  32
asynchronous invocation  28, 30
asynchrony  28
audit logging  3

B
BPE data source  8
BPE database  3
buffer pool  39
  size  18
buffer pool size  18
Business Process container  3
business rule  26
business state machines  26

C
cached data buffer  11
CachedDataBufferSize  38
CEI  14–15
  data store  15
  emitter  9
  event emission  31
  messaging engine  15
  queue  15
Cloudscape  3, 5–6, 15
cluster  8–9
com.ibm.websphere.webservices.http.maxConnection  9
Common Event Infrastructure logging  3
Common Event Interface (CEI)  3
compact phase  20
complex data type names  29
component-managed authentication alias  7
concrete SimpleType  29
concurrency  27, 31
Connection Timeout  10
ConnectionWaitTimeout exception  10
container managed persistence (CMP)  7
contiguity  33
CPU idle  7
CPU idleness  31

D
Dashboard  36
database cross-referencing tables  16
database indexes  17
database log file  16
Datamart database  14
DB2  17
DB2 Automatic Table Maintenance  17
default thread pool  13
  size  8
Deserialization WorkManager  16
Design Advisor  17
development mode  4
DiscardableDataBufferSize  38
driver type  7

ejbCacheSize  38
event arrival rate  13
event delivery  32
event distribution  11
event emitter  14
event monitoring  3, 11
event processing  36
  batch size  15
event server  11
events  31–32
external event table  32

F
full heap GC  20

garbage collection (GC)  19–24
generational collector  19
granularity  27

H
heap size  13–14, 19, 21–23
horizontal scalability  7
HotSpot  19–23
  JVM  20
HTTP transport channel  10
httpd.conf  9
human task  31

I
IBM HTTP Server  9
IBM JVM  34
IBMDEFAULTBP 18
initial heap size 4, 21–22
interruptible process 27
Invoke activity 27

J
Java component 26
Java heap 4, 33–34
fragmentation 33, 35
size 33, 38
Java snippet 26–27
Java Virtual Machine (JVM) 19
Java2 Security 38
JDBC provider 6
JMS queue 32
JVM 30

K
keepAliveEnabled 38

L
large objects 33–35
listener pool 10
listener port 28
live set 21
long-running business processes 27–28
long-running processes 30–31

M
macroflows 30
Max Messages 38
Max Sessions 38
maxConcurrency 7–8
maximum batch size 27
Maximum Concurrent Endpoints 31
Maximum Connections 4, 8, 10
maximum heap size 4, 21, 23
Maximum number of threads 7
maxPersistentRequests 38
MaxSession 10, 28
MaxSpareThreads 9
MDB concurrency 31
message-driven bean (MDB) 8, 28
messaging engine data store 3–5, 7
metadata 29–30
Monitor Dashboard 14
Monitor database 14
Monitor Launchpad 14–15
Monitor messaging engine 15
Monitor Model 14, 36
Monitor Server 14–15, 36
monitoring 3
MQ JMS Non Persistent Listener Port 38
MQ JMS Persistent Listener Port 38
MQ/JMS export binding 28
multiple JVMs 22
multi-threaded SCA clients 27
mySAP.com adapter 35

N
non-interruptible process 27
NumberOfListeners 35
nursery 20
GC 20
size 21

O
ORB 8
ORB thread 8
pool 8
pool size 8
OutOfMemory 12, 22–24
exception 12–13, 24, 33

P
Performance Monitoring Infrastructure (PMI) 3
persistent connections 10
persistent messaging 11
Platform Messaging Component SPI Resource Adapter 7
PMI monitoring 38
poll period 12
poll quantity 12–13
pollPeriod 13, 35
pollQuantity 13, 35
preferred interaction style 28, 30
Pthread 23

Q
Queue connection factory 38

R
rate-trigger heap growth 21
raw partition 39
Redbooks Web site 42
Contact us ix
relationship 16
data source 8
database 3
REORGCHK 18
rollback 15
RUNSTATS 17

S
SCA 8
module 26
synchronous binding 8
schema 5
selectors 26
ServiceManager.locateService() 26
short running flows 31
sib.msgstore.cachedDataBufferSize 11
sib.msgstore.discardableDataBufferSize 11
sib.properties 15
Staff 27
statement cache 39
stderr 20
sweep phase 20
sweeping 20
synchronous event delivery 32
synchronous SCA bindings 26

T
  table space 6, 39
tenured area 20
thread 8
thread pools 31
tracing 3
transaction 27–28
Transactional behavior 27

V
  validateEvent 9
  verbosegc 20, 23, 40
  trace 20, 22–23

W
  Web container 8
  Web service 9
  WebContainer 8
    thread pool 38
  WLM 8
  work manager 7, 13
  WorkArea service 31

X
  XA data source 6
  XSD 29
This IBM Redpaper publication was produced by the IBM WebSphere Process Server, WebSphere Enterprise Service Bus, WebSphere Adapters, and WebSphere Business Monitor performance teams in Austin, Texas; Böblingen, Germany; and Hursley, England. It provides performance tuning tips and best practices based on the performance teams’ experience for WebSphere Process Server 6.0.2, WebSphere Enterprise Service Bus (WebSphere ESB) 6.0.2, WebSphere Adapters 6.0.2, and WebSphere Business Monitor 6.0.2.

If you are either considering or are in the early stages of implementing a solution that incorporates these products, this document should prove to be a useful reference. It describes best practices during application development and deployment and provides a starting point for setup, tuning, and configuration information. It contains a useful introduction to many of the issues that affect these products’ performance and can act as a guide for making rational first choices in terms of configuration and performance settings.

Finally, all of these products build upon the capabilities of the WebSphere Application Server infrastructure. Therefore, solutions that are based on these products will also benefit from existing tuning, configuration, and best practices information for WebSphere Application Server documented in “Related publications” on page 41. You are encouraged to use the information in this paper in conjunction with these references.