First Edition (July 2008)

This edition applies to IBM WebSphere Process Server 6.1.0, IBM WebSphere Enterprise Service Bus 6.1.0, IBM WebSphere Integration Developer 6.1.0, IBM WebSphere Adapters 6.1.0, and IBM WebSphere Business Monitor 6.1.0.

This document created or updated on July 23, 2008.

© Copyright International Business Machines Corporation 2008. All rights reserved.
Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notices</td>
<td>vii</td>
</tr>
<tr>
<td>Trademarks</td>
<td>viii</td>
</tr>
<tr>
<td>Preface</td>
<td>ix</td>
</tr>
<tr>
<td>The team that wrote this paper</td>
<td>x</td>
</tr>
<tr>
<td>Become a published author</td>
<td>xi</td>
</tr>
<tr>
<td>Comments welcome</td>
<td>xi</td>
</tr>
<tr>
<td><strong>Chapter 1. Architecture best practices</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Top 10 tuning and deployment guidelines</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Modeling</td>
<td>3</td>
</tr>
<tr>
<td>1.2.1 Choose non-interruptible over interruptible processes when possible</td>
<td>3</td>
</tr>
<tr>
<td>1.2.2 Choose the appropriate granularity for a process</td>
<td>3</td>
</tr>
<tr>
<td>1.2.3 Use events judiciously</td>
<td>3</td>
</tr>
<tr>
<td>1.2.4 Choose efficient metadata management</td>
<td>4</td>
</tr>
<tr>
<td>1.2.5 Choose business processes over business state machines if possible</td>
<td>5</td>
</tr>
<tr>
<td>1.2.6 Minimize state transitions in business state machines</td>
<td>5</td>
</tr>
<tr>
<td>1.3 Topology</td>
<td>5</td>
</tr>
<tr>
<td>1.3.1 Deploy appropriate hardware</td>
<td>5</td>
</tr>
<tr>
<td>1.3.2 Run production servers in production mode</td>
<td>6</td>
</tr>
<tr>
<td>1.3.3 Use a high performing database (such as DB2)</td>
<td>6</td>
</tr>
<tr>
<td>1.3.4 Deploy local modules in the same server</td>
<td>6</td>
</tr>
<tr>
<td>1.3.5 Best practices for clustering</td>
<td>6</td>
</tr>
<tr>
<td>1.4 Large objects</td>
<td>8</td>
</tr>
<tr>
<td>1.4.1 Factors affecting large object size processing</td>
<td>8</td>
</tr>
<tr>
<td>1.4.2 Large object design patterns</td>
<td>9</td>
</tr>
<tr>
<td>1.5 IBM WebSphere Business Monitor</td>
<td>10</td>
</tr>
<tr>
<td>1.5.1 Event processing</td>
<td>10</td>
</tr>
<tr>
<td>1.5.2 Dashboard</td>
<td>10</td>
</tr>
<tr>
<td>1.5.3 Database server</td>
<td>11</td>
</tr>
<tr>
<td><strong>Chapter 2. Development best practices</strong></td>
<td>13</td>
</tr>
<tr>
<td>2.1 Service Component Architecture considerations</td>
<td>14</td>
</tr>
<tr>
<td>2.1.1 Cache results of ServiceManager.locateService()</td>
<td>14</td>
</tr>
<tr>
<td>2.1.2 Reduce the number of SCA modules, when appropriate</td>
<td>14</td>
</tr>
<tr>
<td>2.1.3 Use synchronous SCA bindings across local modules</td>
<td>14</td>
</tr>
<tr>
<td>2.1.4 Optimize Web service bindings if target is hosted in the same server</td>
<td>14</td>
</tr>
<tr>
<td>2.1.5 Utilize multi-threaded SCA clients to achieve concurrency</td>
<td>15</td>
</tr>
<tr>
<td>2.1.6 Add Quality of Service qualifiers at appropriate level</td>
<td>15</td>
</tr>
<tr>
<td>2.2 Business process considerations</td>
<td>15</td>
</tr>
<tr>
<td>2.2.1 Modeling best practices for a business process or human task</td>
<td>15</td>
</tr>
<tr>
<td>2.2.2 Use one-way invocation styles for long-running processes</td>
<td>15</td>
</tr>
<tr>
<td>2.2.3 Minimize number and size of BPEL variables and business objects</td>
<td>16</td>
</tr>
<tr>
<td>2.3 Transactionality considerations</td>
<td>16</td>
</tr>
<tr>
<td>2.3.1 Exploit SCA transaction qualifiers</td>
<td>16</td>
</tr>
<tr>
<td>2.3.2 Exploit transactional attributes for activities in long-running processes</td>
<td>17</td>
</tr>
<tr>
<td>2.4 Invocation style considerations</td>
<td>18</td>
</tr>
<tr>
<td>2.4.1 Use asynchrony judiciously</td>
<td>18</td>
</tr>
<tr>
<td>2.4.2 Set the preferred interaction style to Sync whenever possible</td>
<td>18</td>
</tr>
</tbody>
</table>
How to get IBM Redbooks publications .................................................. 65
Help from IBM ......................................................................................... 65
Index ........................................................................................................ 67
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

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

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

AIX®
DB2®
developerWorks®
i5/OS®
IBM®
MQSeries®
Redbooks®
Redbooks (logo)®
System i™
Tivoli®
WebSphere®

The following terms are trademarks of other companies:

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

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

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

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, IBM WebSphere Enterprise Service Bus, IBM WebSphere Adapters, and IBM 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 team's experience for the following products:

- WebSphere Process Server 6.1.0
- WebSphere Enterprise Service Bus (WebSphere ESB) 6.1.0
- WebSphere Adapters 6.1.0
- WebSphere Business Monitor 6.1.0

These products represent an integrated development and runtime environment that is based on a key set of service-oriented architecture (SOA) and business process management (BPM) technologies, including Service Component Architecture (SCA), Service Data Object (SDO), and Business Process Execution Language (BPEL) for Web Services. These technologies in turn build upon the core capabilities of the IBM WebSphere Application Server 6.1, including the Java™ Connector Architecture (JCA) V1.5 on which WebSphere Adapters V6.1.0 are based.

This paper discusses the performance implications of the supporting runtime environment, and relates a subset of best practices as well as tuning and configuration parameters for the different software technologies that are involved. The audience for this paper includes a wide variety of groups such as customers, services, technical marketing, and development. Note that this paper is not as comprehensive as a tuning, sizing, or capacity planning guide, although the paper serves as a useful reference for these activities.

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

Finally, because all of these products build upon the capabilities of the WebSphere Application Server infrastructure, solutions based on these products can also benefit from existing tuning, configuration, and best practices information for WebSphere Application Server and corresponding platform JVMs as documented in “Related publications” on page 63. We encourage you to use the information in this paper in conjunction with these references.

The structure of this paper and usage guidelines

We have restructured this paper for better usability since the previous V6.0.2 document. The first three chapters of the paper include best practices and tuning considerations for three different phases of WebSphere Business Process Management (WebSphere BPM) projects as follows:

- Chapter 1, “Architecture best practices” on page 1
- Chapter 2, “Development best practices” on page 13
- Chapter 3, “Performance tuning and configuration” on page 23
In addition, Chapter 1 includes a list of the top 10 tuning and deployment guidelines. We strongly encourage all readers to take note of this list because we have seen numerous instances where this information can be very useful.

Chapter 4, “Initial configuration settings” on page 55 describes configuration options for representative workloads.

The document concludes with a bibliography of useful references (“Related publications” on page 63).

The team that wrote this paper

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

IBM WebSphere Process Server performance team members:
Phani Achanta
Rajiv Arora
Bobbi Brekke
Mike Collins
Gudrun Eckl-Regenhardt
Steve Garde
Jesse Gordon
Jonas Grundler
Weiming Gu
Steve Heuer
Ben Hoflich
Dominik Sebastian Meyer
Thomas Muehlfriedel
Gerhard Pfau
Sefika Prcic
Chris Richardson

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

IBM WebSphere Adapters performance team:
Jimmy DeWitt
Riaz Hussain

IBM WebSphere Business Monitor performance team:
Andy Boothe
Walter Fraser
Kean Kuiper
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® 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
Architecture best practices

This chapter provides guidance about how to architect a high-performing and scalable WebSphere Business Process Management (BPM) solution. It highlights the best practices that are associated specifically with the technologies and features that are delivered in the WebSphere BPM products that we cover in this paper. These products are built on top of existing technologies such as WebSphere Application Server, the default messaging provider, and DB2®. Each of these technologies has associated best practices that apply. However, it is not our intent to enumerate the best practices of these existing technologies in this paper.
1.1 Top 10 tuning and deployment guidelines

This chapter details architectural best practices for all key components. Development best practices, performance tuning, and configuration are covered in subsequent chapters. We strongly encourage you to read all of these chapters, because we have found that this information has been beneficial for numerous customers over the years. However, if you read nothing else in this document, read and adhere to the 10 key tuning and deployment guidelines that we list here, because they are relevant in virtually all performance-sensitive customer engagements:

- Use a high performance disk subsystem. In virtually any realistic topology, a server-class disk subsystem (for example RAID adapter with multiple physical disks) is required on the tiers that host the message and data stores to achieve acceptable performance. We cannot overstate this point. We have seen many cases where the overall performance of a solution has been improved by several factors simply by utilizing appropriate disk subsystems.

- Set an appropriate Java heap size to deliver optimal throughput and response time. JVM™ verbosegc output can help greatly in determining the optimal settings. Further information is available in 3.3.2, “Java tuning parameters” on page 26.

- Where possible, utilize non-interruptible processes (microflows) instead of interruptible processes (macroflows). Macroflows are required for many processes (for example, if human tasks are employed or state needs to be persisted). However, there is significant performance overhead that is associated with macroflows. For details, see 1.2.1, “Choose non-interruptible over interruptible processes when possible” on page 3.

- Use DB2 instead of the default Derby database management system (DBMS). DB2 is a high-performing, industrial strength database that is designed to handle high levels of throughput and concurrency, that scales well, and that delivers excellent response time.

- Tune your database for optimal performance. Proper tuning and deployment choices for databases can greatly increase overall system throughput. For details, see 3.4.8, “Database: General tuning” on page 43.

- Disable tracing. Tracing is clearly important when debugging, but the overhead of tracing severely impacts performance. More information is available in 3.4.1, “Tracing and monitoring considerations” on page 29.

- Configure thread and connection pools for sufficient concurrency. This configuration is especially important for high volume, highly concurrent workloads, because the thread pool settings directly influence how much work the server can process concurrently. For more information, see “Configure thread pool sizes” on page 31.

- Use as few Service Component Architecture (SCA) modules as is practical. Using many modules can produce both additional memory usage and can impact throughput and response time negatively. There are other factors to consider in this decision. For more information, see 2.1.2, “Reduce the number of SCA modules, when appropriate” on page 14.

- Avoid unnecessary usages of asynchronous invocations. Asynchronous invocation is often needed on the edges of modules but not within a module. Use synchronous preferred interaction styles, as described in 2.4.2, “Set the preferred interaction style to Sync whenever possible” on page 18. Also, see 2.4.1, “Use asynchrony judiciously” on page 18 for further guidance about avoiding asynchronous interactions.

- Avoid transaction boundaries that are too granular in SCA and Business Process Execution Language (BPEL). Every transaction commit results in expensive database or messaging operations. Design your transactions with care, as described in 2.3, “Transactionality considerations” on page 16.
1.2 Modeling

This section discusses modeling choices.

1.2.1 Choose non-interruptible over interruptible processes when possible

Use interruptible processes, also known as *macroflows* or *long-running processes*, only when required (for example, long-running service invocations and human tasks). Non-interruptible processes, also known as *microflows* or *short-running processes*, exhibit much better performance at runtime. A non-interruptible process instance is executed in one J2EE™ transaction with no persistence of state, while an interruptible process instance is executed typically in several J2EE transactions, requiring that state be persisted in a database at transaction boundaries.

Whenever possible, use synchronous interactions for non-interruptible processes. A non-interruptible process is much more efficient than an interruptible process.

A process is interruptible if the “Process is long-running” option is selected in the WebSphere Integration Developer Properties view, Details tab for the process.

If interruptible processes are required for some capabilities, separate the processes such that the most frequent scenarios can be executed in non-interruptible processes and exceptional cases are handled in interruptible processes.

1.2.2 Choose the appropriate granularity for a process

A business process and its individual steps need to have *business significance* and should not try to mimic programming level granularity. Use programming techniques such as Plain Old Java Objects (POJOs) or Java snippets for logic without business significance. For more information, see developerWorks article, *Software components: coarse-grained versus fine-grained*, which is available at:


1.2.3 Use events judiciously

The purpose of Common Base Event (CBE) emission in WebSphere Process Server is for business activity monitoring. Because CBE emission uses a persistent mechanism, it is inherently heavy weight. You should use CBE only for events that have business relevance. Further, emitting CBEs to a database is not recommended. Instead, use the messaging infrastructure for CBE emission. Finally, do not confuse business activity monitoring and IT monitoring. The Performance Monitoring Infrastructure (PMI) is far more appropriate for IT monitoring.

With this in mind, the following list generally holds for most customers:

- Customers 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, it is fairly natural to use events to track when long-running activities complete, when human tasks change state, and so forth.

- For short-running flows that complete within seconds, it is usually sufficient to know that a flow completed, perhaps with 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 and end) are usually sufficient for a microflow.
1.2.4 Choose efficient metadata management

In this section, we discuss efficient metadata management choices.

**Follow Java language specification for complex data type names**

While WebSphere Process Server allows characters in business object type names that would not be permissible in Java class names (for example, the underscore "_"), the internal data representation of complex data type names does make use of Java types. As such, performance is better if business object types follow the Java naming standards, because if valid Java naming syntax is used then no additional translation is required.

**Avoid use of anonymous derived types in XML schema definitions**

Some XML Schema Definition (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 declared explicitly, then a new sub-type (a *derived* type) is generated at run time. Performance is generally better if you can avoid using anonymous derived types. So, 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 can utilize that type without degraded performance.

Avoid referencing elements from one XSD in another XSD. Example 1-1 shows an .xsd file that defines an element (AElemnt) and the element referenced by another .xsd file.

**Example 1-1** Referencing an element from another file

If A.xsd defines an element AElement:

```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>
```

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

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

The code in Example 1-1 has shown to perform poorly in our testing. A better choice is to define the type concretely and then make any new elements use this type, as shown in Example 1-2.

**Example 1-2** Revised use of an element

**A.xsd becomes:**

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

**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 example when creating a new business object. It is possible to refer to a business object type by name for instance 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, using `DataObject.getType()`) and reuse that type for the second and future uses.

### 1.2.5 Choose business processes over business state machines if possible

Business state machines provide an attractive way of implementing business flow logic. For some applications, it is more intuitive to model the business logic as a state machine, and the resultant artifacts are easy to understand. Further, WebSphere Process Server 6.1 delivers significant performance improvements for state machines. However, state machines are implemented using the business process infrastructure, so there is always a performance impact when choosing state machines over business processes. If an application can be modeled using either state machines or business processes and performance is a differentiating factor, choose business processes. There are also more options available for optimizing business process performance than there are for state machine performance.

### 1.2.6 Minimize state transitions in business state machines

Where possible, minimize external events to drive state transitions in business state machines. External event driven state transitions are very costly from a performance perspective. In fact, the total time taken to execute a state machine is proportional to the number of state transitions that occur during the life span of the state machine. For example, if a state machine transitions through states $A \rightarrow B \rightarrow B \rightarrow B \rightarrow C$ (four transitions), it is twice as time consuming as making transitions through states $A \rightarrow B \rightarrow C$ (two transitions). Take transitions into consideration when designing a state machine.

Also, automatic state transitions are much less costly than event-driven state transitions.

### 1.3 Topology

In this section we discuss topology considerations.

### 1.3.1 Deploy appropriate hardware

It is very important to pick a hardware configuration that contains the resources that are necessary to achieve high performance in a WebSphere BPM environment. Key considerations in picking a hardware configuration include:

- **Processors**
  
  Ensure that WebSphere Process Server and WebSphere ESB are installed on a modern server system with multiple processors. WebSphere Process Server and WebSphere ESB scale well, both vertically in terms of SMP scaling, and horizontally, in terms of clustering.

- **Memory**
  
  WebSphere Process Server and WebSphere ESB benefit from both a robust memory subsystem as well as an ample amount of physical memory. Ensure that the chosen system has server-class memory controllers and as large as possible L2 and L3 caches (optimally, use a system with at least a 4 MB L3 cache). Make sure there is enough
physical memory for all the applications (JVMs) combined that are expected to run concurrently on the system. 2 GB per JVM is a rough rule of thumb.

- Disk
  Ensure that the systems hosting the messaging data stores and the application and product databases have fast storage, which means using RAID adapters with writeback caches and disk arrays with many physical drives.

- Network
  Ensure that the network is sufficiently fast to not be a system bottleneck. As an example, a dedicated Gigabit Ethernet network is a good choice.

1.3.2 Run production servers in production mode

You can run WebSphere application servers in development mode, which can reduce startup time for the server by using JVM settings to disable bytecode verification and to reduce JIT compilation time. You should not use this setting on production servers, however, because it is not designed to produce optimal runtime performance.

Make sure the “Run in development mode” option for the server is deselected. You can find this setting in the WebSphere administrative console by selecting Servers → Application Servers → server name → Configuration.

You can also create server profiles with production or development templates. Use production profile templates for production servers.

1.3.3 Use a high performing database (such as DB2)

WebSphere Process Server, WebSphere ESB, WebSphere Business Monitor, and WebSphere Adapters are packaged with Derby, an open source database product designed for ease-of-use and platform neutrality. If performance and reliability are important, use an industrial strength database (such as IBM’s DB2) for any performance measurement or production installation. Examples of databases that can be used on DB2 include the BPE database, relationship databases, the messaging engine data stores, adapter event delivery tables, and any adapter specific databases.

1.3.4 Deploy local modules in the same server

If planning to deploy WebSphere Process Server modules on the same physical server, better performance can be achieved by deploying the modules to the same application server JVM, because this allows the server to exploit this locality.

1.3.5 Best practices for clustering

We highly recommend Production Topologies for WebSphere Process Server and WebSphere ESB V6, SG24-7413, which is a comprehensive guide to selecting appropriate topologies for both scalability and high-availability.

This section distills some of the key considerations when trying to scale up a topology for maximum performance.
Use the full support topology for maximum flexibility in scaling

The full support topology prescribes the use of separate clusters for applications, messaging engines, and support applications such as the Common Event Infrastructure (CEI) server, and the Business Rules Manager. As explained in the next section, this topology allows independent control of resources to support the load on each of these elements of the infrastructure. Note as with many system choices, flexibility comes with some cost. For example, synchronous Common Base Event (CBE) emission between an application and the CEI server in this topology is a remote call, which is heavier than a local call. The benefit is the independent ability to scale the application and support cluster. We assume that you are familiar with these kinds of system tradeoffs, because they occur in most server middleware.

The full support topology is described in *Production Topologies for WebSphere Process Server and WebSphere ESB V6*, SG24-7413.

Apply a data-driven scaling methodology

You deploy a clustered topology so that you can add more resources to system components that are bottlenecked due to increasing load. Ideally, it should be possible to scale up a topology arbitrarily to match the required load. The WebSphere Process Server Network Deployment infrastructure provides this capability. However, effective scaling still requires standard performance monitoring and bottleneck analysis techniques to be used.

**Note:** In the list below, we assume that additional cluster members imply additional server hardware.

Considerations include:

- If deploying more than one cluster member (JVM) on a single physical system, it is important to monitor both the resource utilization (CPU, disk, network, and so forth) of the system as a whole and also the utilization by each cluster member. This monitoring allows the detection of a system bottleneck due to a particular cluster member.
- If all members of a cluster are bottlenecked, scaling can be achieved by adding one or more members to the cluster, backed by appropriate physical hardware.
- If a singleton server or cluster member is the bottleneck, additional considerations include:
  - A messaging engine in a cluster with “One of N” policy (to preserve event ordering) can become the bottleneck. Scaling options include:
    - Hosting the active cluster member on a more powerful hardware server, or removing extraneous load from the existing server
    - If the messaging engine (ME) cluster is servicing multiple buses, and messaging traffic is spread across these buses, consider breaking up further to a separate ME cluster per bus.
    - If a particular bus is a bottleneck, consider whether destinations on that bus can tolerate out of order events, in which case you can change the cluster policy to allow workload balancing with partitioned destinations.
  - A database server can become the bottleneck. Approaches to consider are:
    - If the database server is hosting multiple databases that are active (for example, the BPEDB and the MEDB databases), consider hosting each database on a separate server.
    - If a single database is driving load, consider a more powerful database server.
    - Beyond these considerations, database partitioning and clustering capabilities can be exploited.
1.4 Large objects

An issue frequently encountered by field personnel is trying to identify the largest object size that WebSphere Process Server, WebSphere ESB, and the corresponding adapters can effectively and efficiently process. There are a number of factors affecting large object processing in each of these products. We present both a discussion of the issues involved as well as practical guidelines for the current releases of these products.

Note: The single most important factor affecting large object processing is the JVM. There is a significant change in JVM technology from the WebSphere BPM releases V6.0.2 to release V6.1.0. V6.0.2 and earlier products use the 1.4.2 JVM, while V6.1.0 uses Java 5. Thus, this section has been rewritten, and the recommendations and best practices are different than those in previous releases. If you are using WebSphere BPM V6.0.2 or earlier, consult the V6.0.2 performance report.

In general, objects 5 MB or larger can be considered “large” and can require special attention. Objects 100 MB or larger are “very large” and generally require significant tuning to be processed successfully.

1.4.1 Factors affecting 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, the larger the business object that can be processed successfully.

To be able to apply this somewhat general statement, one must first understand that the object size limit is based on three fundamental implementation facts of Java Virtual Machines:

▸ Java heap size limitations
  The limit to the size of the Java heap is operating system dependent. It is not unusual to have a heap size limit of around 1.4 GB for 32-bit JVMs. The heap size limit is much higher on 64-bit JVMs and is typically less of a gating factor on modern hardware configurations than the amount of available physical memory.

▸ Size of in-memory business objects
  Business objects, when represented as Java objects, are much larger in size than when represented in wire format. For example, a business object that consumes 10 MB on an input JMS message queue can result in allocations of up to 90 MB on the Java heap. The reason is that there are many allocations of large and small Java objects as the business object flows through the adapters and WebSphere Process Server or WebSphere ESB.
  There are a number of factors that affect the in-memory expansion of business objects.
  ◾ The single-byte binary wire representation is generally converted to multi-byte character representations (for example Unicode), resulting in an expansion factor of 2.
  ◾ The business object can contain many small elements and attributes, each requiring a few unique Java objects to represent its name, value, and other properties.
  ◾ Every Java object, even the smallest, has a fixed overhead due to an internal object header that is 12-bytes long on most 32-bit JVMs, and larger on 64-bit JVMs.
  ◾ Java objects are padded in order to align on 8-byte or 16-byte address boundaries.
  ◾ As the business object flows through the system, it can be modified or copied, and multiple copies can exist at any given time during the end-to-end transaction. What this
means is that the Java heap must be large enough to host all these business object copies in order for the transaction to complete successfully.

- Number of concurrent objects being processed

The largest object that can be processed successfully is inversely proportional to the number of requests that are processed simultaneously because each request has its own memory usage profile (liveset) as it makes its way through the system. So, simultaneously processing multiple large objects can increase dramatically the amount of memory that is required, because the sum total of each request's livesets has to be able to be fit into the configured heap.

1.4.2 Large object design patterns

In this section, we describe the three proven design patterns to process large object.

Batched inputs: Send large objects as multiple small objects

If a large object needs to be processed, then the solution engineer must find a way to limit the number of large Java objects that are allocated. The primary technique for doing this involves decomposing large business objects into smaller objects and submitting them individually.

If the large objects are actually a collection of small objects, the solution is to group the smaller objects into conglomerate objects less than 1 MB in size. This grouping has been done at a variety of customer sites and has produced good results. 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 have shown that dealing with this complexity is worth the effort as demonstrated by both increased performance and stability.

Note that you can configure certain adapters such as the IBM WebSphere Adapter for Flat Files to use a SplitBySize mode with a SplitCriteria set to the size of each individual object. In this case, a large object is split in chunks of the size that is specified by SplitCriteria to reduce peak memory usage.

Claim check pattern: When a small portion of an input message is used

When the input business object is too large to be carried around in a system and when there are only a few attributes that are actually needed by that process or mediation, you can exploit a pattern called the claim check pattern. The claim check pattern applied to a business object uses the following steps:

1. Detach the data payload from the message.
2. Extract the required attributes into a smaller “control” business object.
3. Persist the larger data payload to a data store and store the “claim check” as a reference in the control business object.
4. Process the smaller control business object, which has a smaller memory footprint.
5. At the point where the solution needs the whole large payload again, check out the large payload from the data store using the key.
6. Delete the large payload from the data store.
7. Merge the attributes in the control business object with the large payload, taking the changed attributes in the control business object into account.

The claim check pattern requires custom code and snippets in the solution. A less developer-intensive variant would be to make use of custom data bindings to generate the control business object. This approach suffers from the disadvantage of being limited to
certain export/import bindings (JMS, MQ, HTTP, and JCA adapters such as Flat Files, E-mail, and FTP) and the full payload still must be allocated in the JVM.

**For very large objects, explore using 64-bit JVMs**
64-bit JVMs do not have an upper limit on the maximum heap size that can be utilized. Given enough heap size, these JVMs can process objects much larger than 32-bit JVMs successfully but at the cost of additional memory and somewhat longer response times.

### 1.5 IBM WebSphere Business Monitor

In this section, we discuss WebSphere Business Monitor components.

#### 1.5.1 Event processing

If a monitor model application is turned off while events are posted to its queue, the number of events in the queue grows. Significant queue depth can impact queue performance negatively in some cases. Therefore, a monitor model should always be running if events are being posted to its queue.

A major factor in event processing performance is the tuning of the WebSphere Business Monitor database. You should pay attention especially to adequate bufferpool sizes to minimize disk reading activity and the placement of the database logs, which ideally are on a physically separate disk subsystem from the database table spaces.

Another factor in event processing performance is the configuration of the integration bus that WebSphere Business Monitor uses for creating the destination queues for each installed model. The best performance can be obtained by having the service integration bus be present on the same WebSphere node and server as the WebSphere Business Monitor server. In addition, configure the messaging engine for this bus to use a DB2 data store for its message store instead of a file store. It is important to also tune this database by having the tables and logs on separate high performance disk subsystems.

#### 1.5.2 Dashboard

The platform requirements of the Dashboard/Alphablox/Portal stack are relatively modest compared to those of WebSphere Business Monitor server and the database server. The most important consideration for good Dashboard performance is to size and configure the database server correctly. Be sure it has enough CPU capacity for anticipated data mining queries, enough RAM for bufferpools and so forth, and plenty of disk arms. Optimally, store the databases for WebSphere Business Monitor, Datamart, and each of their logs, on separate RAID arrays.

Also, the design of the business process being monitored can have a significant affect on Dashboard performance. Stopwatch-style measures in the business process can have a particularly negative effect on performance.
1.5.3 Database server

Both event processing and Dashboard rely on a fast, well-tuned database server for good performance. The design of WebSphere Business Monitor assumes that any customer using it has strong on-site database administrator skills. We strongly advise that you read and follow the database tuning advice and recommendations beginning in 3.4.8, “Database: General tuning” on page 43.
Development best practices

This chapter discusses best practices that are relevant to the solution developer. It primarily addresses modeling, design, and development choices that are made while designing and implementing a WebSphere BPM solution. The WebSphere Integration Developer tool is used to implement the vast majority of these best practices.
2.1 Service Component Architecture considerations

This section covers the Service Component Architecture (SCA) considerations.

2.1.1 Cache results of ServiceManager.locateService()

When writing Java code to locate an 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 Reduce the number of SCA modules, when appropriate

WebSphere Process Server components are assembled into modules for deployment. When assembling modules, we recognize that many factors come into play. Performance is one key factor, but maintainability, versioning requirements, and module ownership must be considered as well. In addition, more modules can allow for better distribution across servers and nodes. Still, it is important to recognize that modularization also has a cost. When components will be placed together in a single server instance, it is best to package them within a single module for best performance.

2.1.3 Use synchronous SCA bindings across local modules

For cross-module invocations, where the modules are likely to be deployed locally, that is within the same server JVM, we recommend using the synchronous SCA binding. This binding is optimized for module locality and outperforms other bindings. Note that synchronous SCA is as expensive as other bindings when invocations are made between modules located in different WebSphere Process Server or WebSphere Enterprise Service Bus (WebSphere ESB) servers.

2.1.4 Optimize Web service bindings if target is hosted in the same server

If the target of a Web service import binding is hosted locally in the same application server, the performance can be further improved by exploiting the optimized communication path provided by the Web container. Normally requests from the Web service clients are sent through the network connection between the client and the service provider. For local Web service calls, however, WebSphere Application Server offers a direct communication channel bypassing the network layer completely. To optimize the Web container:

- Set the `enableInProcessConnections` custom property for the Web container to `true`. You can find this property in the administrative console by selecting Application servers → server name → Container Settings → Web Container Settings → Web container → Additional Properties → Custom Properties.
- Do not use a wildcard character (*) for the host name of the Web container port. Replace it with the host name or IP address. You can find this property in the administrative console by selecting Application servers → server name → Container Settings → Web Container Settings → Web container → Additional Properties → Web container transport chains → WCInboundDefault → TCP inbound channel (TCP_2) → Related Items → Ports → WC_defaulthost → Host.
- Use localhost instead of the host name in the Web service client binding. If the actual host name is used, and even if it is aliased to localhost, this optimization will be disabled. You can find this property in the administrative console by selecting Enterprise
Applications → application name → Manage Modules → application EJB™ jar →
Web services client bindings → Preferred port mappings → binding name. Use
localhost (for example localhost:9080) in the URL.

- Make sure there is not an entry for your server host name and IP address in your server’s
hosts file for name resolution. An entry in the hosts file inhibits this optimization by adding
name resolution overhead.

2.1.5 Utilize multi-threaded SCA clients to achieve concurrency

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

2.1.6 Add Quality of Service qualifiers at appropriate level

Quality of Service (QoS) qualifiers such as business object instance validation can be added
at the interface level, or at an operation level within an interface. Because there is additional
overhead associated with QoS qualifiers, do not apply a qualifier at the interface level if it is
not needed for all operations of the interface.

2.2 Business process considerations

In this section, we discuss various business process considerations.

2.2.1 Modeling best practices for a business process or human task

Key modeling best practices for a business process or human tasks are:

- Use the Audit Logging property for business processes only if you need to log events in
the BPE database. This property can be set at the activity or process level. If set at the
process level the setting is inherited by all activities.

- To improve human task performance, use group work items for large groups (staff verb:
Group).

- For long-running processes, disable the Enable persistence and queries of
business-relevant data flag In the Properties view, Server tab for both the process and
for each individual BPEL activity. Enabling this flag causes details and history of the
execution of this activity to be stored in the BPC database. This storage increases the load
on the database and the amount of data that is stored for each process instance. You
should use this setting only if this specific information needs to be retrieved later.

2.2.2 Use one-way invocation styles for long-running processes

When designing long-running process components, ensure that the interface does not use
two-way (request-response) synchronous semantics, as this ties up the caller’s resources
(thread, transaction, and so forth) until the process completes. For long-running processes,
this is highly undesirable. Instead, such processes should either be invoked asynchronously,
or using a one-way synchronous call, where no response is expected.

In addition to these performance considerations, there are also issues with recoverability.
Suppose a non-interruptible process that was started using a persistent JMS message calls a
long-running process using the two-way request-response semantics. If the server fails after
the long-running process has completed, but before the caller's transaction is committed,
upon server restart the caller's transaction is rolled back and then retried. However, the result
of the execution of the long-running process on the server is not rolled back, because it was
committed before the server failure. As a result, the long-running process on the server is
executed twice. This duplication causes functional problems in the application unless
corrected manually.

2.2.3 Minimize number and size of BPEL variables and business objects

Considerations to minimize the number and size of BPEL variables and business objects
include:

- Use as few variables as possible, and minimize the size and the number of 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
  save very costly. Smaller business objects are also more efficient to process when
  emitting monitor events.
- Specify variables as Data Type variables to improve runtime performance.
- Use transformations (maps or assigns) to produce smaller business objects by mapping
  only fields that are necessary for the business logic.

2.3 Transactionality considerations

One of the strengths of the WebSphere Process Server platform is the precise control that it
provides for specifying transactional behavior. We strongly recommend that when modeling a
process or mediation assembly, the modeler carefully design the desired transaction
boundaries as dictated by the application's needs. Transaction boundaries are expensive in
system resources. Thus, this section guides the modeler in how to avoid unnecessary
transaction boundaries.

The following general guiding principles are at work here:

- The throughput of a particular usage scenario is inversely related to the number of
  transaction boundaries that are traversed in the scenario, so fewer transactions is faster.
- In user-driven scenarios, improving response time can require more granular transaction
  boundaries, even at the cost of throughput.
- Transactions can span across synchronous invocations but cannot span asynchronous
  invocations.

We provide more detail in the following sections.

2.3.1 Exploit SCA transaction qualifiers

In an SCA module assembly, the number of transaction boundaries can be reduced by
allowing transactions to propagate across components. For any pair of components where
this reduction is desired, we recommend using the following golden path:

- SuspendTransaction= false, for the calling component's reference
- joinTransaction= true, for the called component's interface
- Transaction= any|global, for the implementation of both components
These settings assume that the first component in such a chain either starts or participates in a global transaction.

2.3.2 Exploit transactional attributes for activities in long-running processes

Although SCA qualifiers control component-level transactional behavior, there are additional transactional considerations in long-running processes that can cause activities to run in multiple transactions. You can change the scope of those transactions and the number of transactions with the transactional behavior settings on Java Snippet, Human Task, and Invoke activities. Use `preferredInteractionStyle=synchronous` for components that do not span multiple transactions.

For a detailed description of these settings, see the WebSphere Process Server information center, which is available at:


There are four possible settings:

- Commit before
- Commit after
- Participates
- Requires own

Only the Participates setting does not require a new transaction boundary. The other three settings require the process flow container to start a new transaction before executing the activity, after executing the activity, or both before and after.

In general, the Participates attribute provides the best throughput, and you should use it wherever possible. This is true for both synchronous and asynchronous activities. In the two-way asynchronous case, it is important to understand that the calling transaction always commits after sending the request. The Participates setting refers to the transaction that is started by the process engine for the response. When set, this allows the next activity to continue on the same transaction.

In special cases, the other transaction settings might be preferable (explained in more detail in the information center):

- Use Commit before in parallel activities that start new branches to ensure parallelism. As noted in the information center, there are other constraints that you need to consider.
- Use Commit after for inline human tasks in order to increase responsiveness to human users. When a human user issues a task completion, the thread/transaction that handles the action is used to resume navigation of the Human Task activity in the process flow. The user's task completion action does not complete until the process engine commits the transaction. If the Participates setting is used, the commit is delayed, which forces a longer response time to the user. This is a classic response time versus throughput trade-off.
2.4 Invocation style considerations

This section discusses invocation style considerations.

2.4.1 Use asynchrony judiciously

You can wire components and modules to each other either synchronously or asynchronously. The choice of interaction style can have a profound impact on performance. Thus, you need to exercise care when making this choice.

2.4.2 Set the preferred interaction style to Sync whenever possible

Many WebSphere Process Server component types, such as interface maps or business rules, invoke their target components based on the target interface’s setting of preferred interaction style. Because synchronous cross-component invocations are better performing, it is recommended to set the 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 you set this to Async.

By default, the Preferred Interaction Style is set to Any, meaning that it is determined by the caller’s context. If the caller is a long-running process, a Preferred Interaction Style setting of Any is treated as asynchronous. If the caller is a non-interruptible business flow, a Preferred Interaction Style setting of any is treated as synchronous.

The invocation logic of processes is explained in more detail in the WebSphere Process Server information center at:


Some additional considerations are:

- When setting an interface's Preferred Interaction Style setting to Async, it is important to realize the downstream implications. Any components invoked downstream inherit the Async interaction style unless they explicitly set their Preferred Interaction Style to Sync.
- At the input boundary to a module, exports that represent asynchronous transports such as MQ, JMS, or JCA (with async delivery set) set the interaction style to Async, which can cause downstream invocations to be Async if the Preferred Interaction Style is left at Any.
- For an SCA import, you can use its Preferred Interaction Style to specify whether the cross-module call should be Sync or Async.
- For other imports that represent asynchronous transports such as MQ or JMS, it is not necessary to set the Preferred Interaction Style to Async. Doing so introduces an unnecessary async hop between the calling module and the invocation of the transport.

2.4.3 Avoid cross-component asynchronous invocations within a module

It is important to realize that asynchronous invocations are intended to provide a rich set of qualities of service, including transactions, persistence, and recoverability. Thus, think of an asynchronous invocation as a full messaging “hop” to its target. When the intended target of the invocation is in the same module, a synchronous invocation can yield much higher performance.
2.5 Messaging considerations

This section discusses messaging considerations.

2.5.1 Choose a MQ or MQ JMS binding rather than MQ link

Prior to the WebSphere Process Server and WebSphere ESB V6.0.2 release, an MQ link configuration was required to use WebSphere MQ queues directly in conjunction with WebSphere Process Server and WebSphere ESB. However, the MQ and MQ JMS bindings (added in the V6.0.2 release) are easier to configure and deliver much better performance.

2.5.2 Set MaxSession for the MQ JMS export binding

When an export with an MQ JMS binding is created in WebSphere Integration Developer, the “maximum number of sessions” property is set to 1, by going to the Properties view. Then, go to the Binding tab and the End-point configuration tab, in the Listener Port Properties section. MaxSession informs the container how many concurrent incoming MQ JMS requests can be processed at the same time. For better concurrency, you need to change this property to a larger value, for example 10.

2.5.3 Use mediations that benefit from WebSphere ESB optimizations

Certain types of mediations benefit from internal optimization in WebSphere ESB and deliver improved performance. This specialized optimization can be regarded as a kind of fast path through the code and is in addition to any general optimization of the WebSphere ESB mediation code.

There are three categories of mediation primitives in WebSphere ESB that benefit to a greater or lesser degree from the internal optimizations:

- Category 1 (greatest benefit)
  - Route on Message Header (Message Filter primitive)
  - XSLT primitive (transforming on /body as the root)
  - Endpoint Lookup primitive without Xpath user properties
  - Event Emitter primitive (CBE header only)

- Category 2 (medium benefit)
  - Route on Message Body (Message Filter primitive)

- Category 3 (lowest benefit)
  - Custom Mediation primitive
  - Database Lookup primitive
  - Message Element Setter primitive
  - Business Object Map primitive
  - Fan Out primitive
  - Fan In primitive
  - Set Message Type primitive
  - Message Logger primitive
  - Event Emitter primitive (Except for CBE Header only)
  - Endpoint Lookup primitive utilizing Xpath user properties
  - XSLT primitive (with a non /body root)
There is, therefore, an ideal pattern of usage in which these mediation primitives can take advantage of a fastpath through the code. Fully fastpathed flows can contain any of the mediation primitives in category 1, for example:

→ XSLT Primitive(/body) → Route On Header → EndPointLookup (non-Xpath) →

Partially fastpathed flows can contain a route on body filter primitive (category 2) and any number of category 1 primitives, for example:

→ XSLT Primitive(/body) → Route on body →

In addition to these optimizations, the ordering of primitives can be important. If the mediation flow contains an XSLT primitive (with a root of /body, that is the category 1 variant) and category 3 primitives, then place the XSLT primitive ahead of the other primitives. Thus, the first order listed here is preferable to the second order listed:

→ Route On Header → XSLT primitive(/body) → Custom Mediation primitive →

→ Route On Header → Custom Mediation primitive → XSLT primitive(/body) →

Understand that there are costs that are associated with any primitive regardless of whether the flow is optimally configured. If an Event Emitter primitive is using event distribution or a Message Logger primitive is included, there are associated infrastructure overheads for such remote communications. Large messages increase processing requirements proportionally for primitives (especially those accessing the body), and a custom primitive can contain code that might not be optimally written. These guidelines can help in designing for performance, but they cannot guarantee speed.

Benefits from the optimizations are predictably achieved using WebSphere Integration Developer generated WSDL. Some types of WSDL, such as RPC-Encoded, might not benefit. WSDL should be Doc/Lit Wrapped to benefit from these optimizations.

### 2.6 Large object best practices

This section discusses large object best practices.

#### 2.6.1 Avoid lazy cleanup of resources

Lazy cleanup of resources adds to the liveset that is required when processing large objects. You need to clean up any resources that can be cleaned up (for example, by dropping object references when no longer required) as soon as is practical.

#### 2.6.2 Avoid tracing when processing large business objects

Tracing and logging can add significant memory overhead. A typical tracing activity is to dump the business object payload. Creating a string representation of a large business object can trigger allocation of many large and small Java objects in the Java heap. Avoid turning on tracing when processing large business object payloads in production environments.
2.6.3 Avoid buffer-doubling code

Study the memory implications of using Java data structures which expand their capacity based on input (for example StringBuffer, ByteArrayOutputStream). Such data structures usually double their capacity when they run out of space. This doubling can produce significant memory pressure when processing large objects. If possible, always assign an initial size to such data structures.

2.6.4 Make use of deferred-parsing friendly mediations for XML docs

Certain mediations can reduce memory pressure as they retain the document in their native form and avoid inflating them into their full business object representation. These mediations are listed in 2.5.3, “Use mediations that benefit from WebSphere ESB optimizations” on page 19. Where possible, use these mediations.

2.7 Adapters: configure synchronous event delivery

By default, event delivery for inbound messages from adapters is done 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, it is recommended that you use synchronous semantics for better performance. This is done in WebSphere Integration Developer tooling, and cannot be changed at deployment time.

To set Event Delivery to use synchronous semantics in WebSphere Integration Developer 6.1 perform the following steps:
1. Change to Business Integration perspective and open the appropriate assembly diagram.
2. Right-click the adapter export component and select Show in Properties.
3. Select the Performance attributes section under the Binding tab.
4. Change the value in the Interaction Style pull-down menu to Sync.
5. Save your changes, rebuild the application, export the EAR file, and redeploy.
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 when performing system tuning for the WebSphere Business Process Management (WebSphere BPM) products and also other products in the system (for example DB2). The documentation for each of these products contains a wealth of information regarding performance, capacity planning, and configuration. This documentation offers the best guidance for performance considerations in a variety of operational environments. Assuming that you have addressed all of these issues from the perspective of the actual product, additional levels of performance implications are introduced at the interface between these products and the products that we cover in this paper.

A number of configuration parameters are available to the system administrator. While this chapter identifies several specific parameters 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.

In the first section of this chapter, we describe a methodology to use when tuning a deployed system. Then we provide a basic tuning checklist that enumerates the major components and their associated tuning concepts. The subsections that follow address tuning in more detail, first describing several tuning parameters and their suggested setting (where appropriate), and finally providing advanced tuning guidelines for more detailed guidance for key areas of the system. While there is no guarantee that following the guidance in this chapter will provide immediately acceptable performance, it is likely that you can expect degraded performance if these parameters are set incorrectly.

See “Related publications” on page 63 for more information about related documentation that might prove valuable when tuning a particular configuration.
3.1 Performance tuning methodology

We recommend a system-wide approach to performance tuning of a WebSphere BPM environment. Note that we do not describe exhaustively the art of system performance tuning, which requires training and experience. Rather, we highlight some key aspects of tuning that are particularly important.

It is important to note that tuning encompasses every element of the deployment topology:

- Physical hardware topology choices
- Operating system parameter tuning
- WebSphere Process Server, WebSphere Application Server, and message engine tuning

The methodology for tuning can be stated very simply as an iterative loop:

1. Pick a set of reasonable initial parameter settings.
2. Run the system.
3. Monitor the system to obtain metrics that indicate whether performance is being limited.
4. Use monitoring data to guide further tuning changes.
5. Repeat this process until you are satisfied with the performance levels.

Pick a set of reasonable initial parameter settings as follows:

- Use the tuning checklist in 3.2, “Tuning checklist” on page 25 for a systematic method for setting parameters.
- For a set of initial values, consult Chapter 4, “Initial configuration settings” on page 55.
- Monitor the system. We recommend monitoring the system (or systems) to determine system health, as well as to determine the need for further tuning. Monitor the following:
  - For each physical machine in the topology, including front-end and back-end servers such as Web servers, and database servers, monitor the following utilization using relevant operating system tools such as `vmstat`, `iostat`, `netstat`, or an equivalent:
    - CPU utilization
    - Memory utilization
    - Disk utilization
    - Network utilization
  - For each JVM process that is started on a physical machine, that is WebSphere Process Server server, messaging engine server, and so forth:
    - Use tools such as `ps` or an equivalent to obtain the CPU and memory usage per process
    - Collect `verbosegc` statistics
  - For each WebSphere Process Server or messaging engine JVM, use IBM Tivoli® Performance Viewer to monitor:
    - For each data source, the data connection pool utilization
    - For each thread pool (webcontainer, default, and work managers), the thread pool utilization

Use monitoring data to guide further tuning changes is a vast topic that requires skill and experience. In general, this phase of tuning requires the analyst to look at the collected monitoring data, detect performance bottlenecks, and do further tuning. The key characteristic about this phase of tuning is that it is driven by the monitoring data that is collected in the previous phase.
Examples of performance bottlenecks include, but are not limited to:

- Excessive utilization of physical resources such as CPU, disk, memory, and so forth. You can resolve this type of issue either by adding more physical resources or by rebalancing the load more evenly across the available resources.
- Excessive utilization of virtual resources. Examples include heap memory, connection pools, thread pools, and so forth. For this type of issue, you need to use tuning parameters to remove the bottlenecks.

### 3.2 Tuning checklist

This checklist serves as a guide or a “to do” list when tuning a WebSphere BPM solution. We cover each of these topics in more detail in the remainder of this chapter.

#### Common
- Disable tracing and monitoring when possible
- Move databases from the default Derby to a high performance DBMS such as DB2
- When security is required (generally the case), use administration and application security instead of Java2 security to minimize overhead.
- Use appropriate hardware configuration for performance measurement, for example ThinkPads and desktops are not appropriate for realistic performance evaluations.
- Do not run production server in development mode or with a development profile
- Do not use the Unit Test Environment (UTE) for performance measurement
- Configure the message-driven bean (MDB) activation specifications
- Configure for clustering (where applicable)
- Configure thread pool sizes
- Configure data sources: connection pool size, prepared statement cache size
- Disable validation in Common Event Infrastructure (CEI) Emitter

#### Messaging and message bindings
- Optimize the activation specification (JMS)
- Optimize the queue connection factory (JMS)
- Configure connection pool size (JMS)
- Optimize listener port configuration (MQ JMS, MQ)
- Configure the service integration bus data buffer sizes

#### Database
- Place database table spaces and logs on a fast disk subsystem
- Place logs on a separate device from the table space containers
- Maintain current indexes on the tables
- Update database statistics
- Set log file sizes correctly
- Optimize the buffer pool size (DB2) or buffer cache size (Oracle®)

#### Java
- Set the heap / nursery sizes to manage memory efficiently
– Choose the appropriate garbage collection policy

► WebSphere Adapters
  – Configure pollPeriod and pollQuantity settings
  – Configure the application server thread pool
  – Configure the work manager thread pool

► Monitor
  – Configure CEI
  – Set message consumption batch size

### 3.3 Tuning parameters

This section lists performance tuning parameters commonly used in tuning the products covered in this paper. Some flags or check boxes are common to all or a subset of the products, while others are specific to a particular product. Unless stated otherwise, you set all of these parameters using the administrative console.

#### 3.3.1 Tracing and logging flags

Tracing and logging are often necessary when setting up a system or debugging issues. However, these capabilities produce performance overhead that is often significant. Thus, you need to minimize their use when evaluating performance or in production environments.

To enable or disable tracing, go to Troubleshooting → Logs and Trace → server name → Change Log Detail Levels and set both the Configuration and Runtime to *=all=disabled*.

To change the PMI level go to Monitoring and Tuning → Performance Monitoring Infrastructure → server name and select none.

#### 3.3.2 Java tuning parameters

In this section, we list a few frequently used Java Virtual Machine (JVM) tuning parameters. For a complete list, consult the JVM tuning guide that is offered by the JVM supplier.

You can access the JVM administrative panel from Servers → Application Servers → server name → Server Infrastructure → Java and Process Management → Process Definition → Additional Properties → Java Virtual Machine.

**Java garbage collection policy**

The default garbage collection algorithm on platforms with an IBM JVM is Mark-Sweep-Compact. In some cases, the generational concurrent (**gencon**) algorithm delivers better performance with a tuned nursery size as discussed in the next section. To change the GC policy to gencon, add `-Xgcpolicy:gencon` to the Generic JVM arguments on the Java Virtual Machine administrative panel.

**Java heap sizes**

To change the default Java heap sizes, set the initial heap size and maximum heap size settings explicitly on the Java Virtual Machine administrative panel.
If Generational Concurrent Garbage Collector is used, the Java heap is divided into a new area (nursery) where new objects are allocated and an old area (tenured space) where longer lived objects reside. The total heap size is the sum of the new area and the tenured space. The new area size can be set independently from the total heap size. Typically, the new area size should be set between ¼ and ½ of the total heap size. The relevant parameters are:

- `-Xmns<size>`: Initial new area size
- `-Xmnx<size>`: Maximum new area size
- `-Xmn<size>`: Fixed new area size

### 3.3.3 MDB activation specification

There are a few shortcuts to access the MDB activation specification tuning parameters, as follows:

- Resources → Resource Adapters → J2C activation specifications → *ActivationSpec name*
- Resources → JMS → Activation specifications → *ActivationSpec name*
- Resources → Resource Adapters → Resource adapters → resource adapter name → Additional properties → J2C activation specifications → *ActivationSpec name*

Two custom properties in the MDB activation specification have considerable performance implications:

- `maxConcurrency`
- `maxBatchSize`

We discuss these properties further in “Tune MDB activation specification properties” on page 31.

### 3.3.4 MQ listener port

For the MQ or MQ JMS binding, a listener port is used for configuring the delivery of inbound messages to WebSphere Process Server or WebSphere Enterprise Service Bus (WebSphere ESB). The Maximum sessions property of the listener port has performance impact. The property can be accessed by Servers → Application servers → server name → Communications → Messaging → Messaging Listener Service → Additional Properties → Listener Ports → *listener port name*.

### 3.3.5 Thread pool sizes

WebSphere uses thread pools to manage concurrent tasks. You can set the Maximum Size property of a thread pool under Servers → Application servers → server name → Additional Properties → Thread Pools → thread pool name.

Typically, you need to tune the following thread pools:

- Default
- ORB.thread.pool
- WebContainer
### 3.3.6 JMS connection pool sizes

There are a few ways of accessing the JMS connection factories and JMS queue connection factories from WebSphere administrative console, as follows:

- **Resources → Resource Adapters → J2C connection factories → factory name**
- **Resources → JMS → Connection factories → factory name**
- **Resources → JMS → Queue connection factories → factory name**
- **Resources → Resource Adapters → Resource adapters → resource adapter name** (for example SIB JMS Resource Adapter) → Additional properties → J2C connection factories → factory name

From the connection factory panel, open **Additional Properties → Connection pool properties**. Set the **Maximum Connections** property for the maximum size of the connection pool.

### 3.3.7 Data source connection pool size

You can access data sources from:

- **Resources → JDBC™ → Data sources → data source name**
- **Resources → JDBC Providers → JDBC provider name → Additional Properties → Data sources → data source name**

The maximum size of the data source connection pool is limited by the value of Maximum Connections property, which you can configure from the data source panel's **Additional Properties → Connection pool properties**.

Typically, you need to tune the following data sources:

- BPEDataSource for the BPE database
- SCA application bus messaging engine data source
- SCA system bus messaging engine data source
- CEI bus messaging engine data source

### 3.3.8 Data source prepared statement cache size

You can configure the data source prepared statement cache size from the data source's **Additional properties → WebSphere Application Server data source properties**.

The BPC container in WebSphere Process Server uses prepared statements extensively. The statement cache size of this data source should be at least 128.

### 3.3.9 Messaging engine properties

Two message engine custom properties can impact the messaging engine performance:

- **DiscardableDataBufferSize**
- **CachedDataBufferSize**

You can access the properties under **Service Integration → Buses → bus name → Messaging Engines → messaging engine name → Additional properties → Custom properties**.
3.4 Advanced tuning

This section discusses advanced tuning options.

3.4.1 Tracing and monitoring considerations

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 BPM product set provides rich monitoring capabilities, both in terms of business monitoring through the Common Event Interface (CEI) and audit logging, and system performance monitoring through the Performance Monitoring Infrastructure (PMI) and the Application Response Measurement (ARM) infrastructure. Although these capabilities provide insight into the performance of the running solution, these features can degrade overall system performance and throughput.

**Important:** It is recommended that you use tracing and monitoring judiciously and, when possible, that you turn off tracing and monitoring entirely to ensure optimal performance.

Most tracing and monitoring is controlled using the administrative console. Validate that the appropriate level of tracing and monitoring is set for PMI Monitoring, Logging, and Tracing using the administrative console.

Further, use the administrative console to validate that the “Audit logging” and “Common Event Infrastructure logging” options are disabled in the Business Process container, unless these capabilities are required for business reasons.

The 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.

**Configure the work area 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 then deserializing the work area from the reply. The default work area service as well as the work area partition service makes use of 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 as it circumvents the costly checking of the outbound or incoming work area size.

**Disable the CEI emitter validateEvent property**

The default behavior of CEI event emitters is to validate the events to ensure that all fields are correctly filled in. This is unnecessary for events generated by the WebSphere Process Server and WebSphere ESB systems. To disable event validation, a custom property needs to be added to the emitter factory profile using:

- Resources → CEI Infrastructure Provider (cell scope) → Emitter Factory Profile → Default Common Event Infrastructure emitter → Custom Properties

Add a new property with name `validateEvent` with value set to `false`.
3.4.2 Tuning for large objects

This section discusses tuning for large objects.

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

A key factor that affects large object processing is the maximum size of the Java heap. In this section, we discuss how to set the heap size as big as possible on two commonly used platforms. For more comprehensive heap setting techniques, consult 3.4.11, “Advanced Java heap tuning” on page 48.

- **Windows®:**
  
  Due to address space limitations in the Windows 32-bit operating system, the largest heap that can be obtained is around 1.4 GB to 1.6 GB for 32-bit JVMs. When using a 64-bit Windows JVM, however, the heap size is only limited by the available physical memory.

- **AIX®:**
  
  Using the normal Java heap settings, the Java5 JVM supports heaps 2 GB to 2.4 GB on 32-bit systems. Note that because the 4 GB address space allowed by the 32-bit system is shared with other resources, the actual limit of the heap size depends on memory usage by resources such as thread stacks, JIT compiled code, loaded classes, shared libraries, buffers used by OS system services, and so forth. An extremely large heap squeezes address space that is reserved for other resources and can cause runtime failures. On 64-bit systems, the available address space is practically unlimited, so the heap size is usually limited only by available memory.

**Reduce or eliminate other processing while processing a large object**

One way to allow for larger object sizes is to limit the concurrent processing within the JVM. One should not expect to be able to process a steady stream of the largest objects possible concurrently with other WebSphere Process Server, WebSphere ESB, and WebSphere Adapters activities. The operational assumption that you need to make when considering large objects is that not all objects are “large” or “very large” and that large objects will not arrive very often, perhaps once or twice per day. If more than one “very large” object is processed concurrently, the likelihood of failure increases dramatically.

The size and number of the normally arriving smaller objects can affect the amount of Java heap memory consumption in the system. Generally speaking, the heavier the load on a system when a large object is being processed the more likely that 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 10 seconds) 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 “Configure the poll period and poll quantity” on page 38.

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.
3.4.3 Tuning for maximum concurrency

For most high volume deployments on server-class hardware, there are many operations that take place simultaneously. Tuning for maximum concurrency ensures that the server accepts enough load to saturate its CPUs. A sign of an inadequately tuned configuration is when additional load does not result in additional CPU utilization, while the CPUs are not fully utilized. To optimize these operations for maximum concurrency, the general guideline is to follow the execution flow and remove bottlenecks one at a time. Note that higher concurrent processing means higher resource requirements (memory and number of threads) on the server. It needs to be balanced with other tuning objectives, such as the handling of large objects, handling large numbers of users, and providing a good response time.

Tune Edge components for concurrency

The first step to tuning for maximum concurrency is to ensure that business objects are handled concurrently at the edge components of WebSphere Process Server or WebSphere ESB. If the input business objects come from the adapter, ensure the adapter is tuned for concurrent delivery of input messages. See 3.4.6, “WebSphere Adapters tuning” on page 38 for more details on tuning adapters.

If the input business objects come from Web services export binding or direct invocation from JSPs or servlets, make sure the WebContainer thread pool is right sized. To allow for 100 in-flight requests handled concurrently by the WebSphere Process Server, the maximum size of the WebContainer thread pool needs to be set to 100 or larger.

If the input business objects come from the messaging, you need to tune the activation specification (MDB bindings) and Listener ports (MQ or MQJMS bindings).

Tune MDB activation specification properties

For each JMS export component, there is an MDB and its corresponding activation specification (JNDI name: module name and export component name_AS). The default value for maxConcurrency of the JMS export MDB is 10, meaning up to 10 business objects from the JMS queue can be delivered to the MDB threads concurrently. Change it to 100 if a concurrency of 100 is desired.

The maximum batch size in the activation spec also has an impact on performance. The default value is 1. The maximum batch size value determines how many messages are taken from the messaging layer and delivered to the application layer in a single step (note that this does NOT mean that this work is done within a single transaction, and therefore this setting does not influence transactional scope). Increasing this value, for example to 8, for activation specs associated with long-running business processes can improve performance.

Configure the listener port

If the MQ or MQ/JMS bindings are used, a listener port is used to configure the delivery of inbound messages to WebSphere Process Server or WebSphere ESB. In this case, the Maximum sessions property serves an identical purpose as the maxConcurrency parameter in the activation specification for JMS bindings. Increase this value to an appropriate value.

A listener port is created when the SCA module containing the MQ or MQJMS binding deploys to the server.

Configure thread pool sizes

The sizes of thread pools have a direct impact on a server's ability to run applications concurrently. For maximum concurrency, the thread pool sizes need to be set to optimal values. Increasing the maxConcurrency or Maximum sessions parameters only enables the
concurrent delivery of business objects from the JMS or MQ queues. In order for the WebSphere Process Server or WebSphere ESB server to process multiple requests concurrently, it is also necessary to increase the corresponding thread pool sizes to allow higher concurrent execution of these Message Driven Beans (MDB) threads.

MDB work is dispatched to threads allocated from the default thread pool. Note that all MDBs in the application server share this thread pool, unless a different thread pool is specified. This means that the default thread pool size needs to be larger, probably significantly larger, than the maxConcurrency of any individual MDB.

Threads in the Web container thread pool are used for handling incoming HTTP and Web Services requests. Again, this thread pool is shared by all applications deployed on the server. As discussed earlier, it needs to be tuned, likely to a higher value than the default.

ORB thread pool threads are employed for running ORB requests, for example remote EJB calls. The thread pool size needs to be large enough to handle requests coming in through EJB interface, such as certain human task manager APIs.

**Configure dedicated thread pools for MDBs**
The default thread pool is shared by many WebSphere Application Server tasks. It is sometimes desirable to separate the execution of JMS MDBs to a dedicated thread pool. Follow these steps to change the thread pool used for JMS MDB threads:

1. Create a new thread pool, say MDBThreadPool, on the server.
2. Open the Service Integration Bus (SIB) JMS Resource Adapter administrative panel with server scope from Resources → Resource Adapters → Resource adapters. If the adapter is not shown, go to Preferences, and select Show built-in resources.
3. Change Thread pool alias from Default to MDBThreadPool.
4. Repeat steps 2 and 3 for SIB JMS Resource Adapters with node and cell scope.
5. Restart the server for the change to be effective.

SCA module MDBs for asynchronous SCA calls use a separate resource adapter, the Platform Messaging Component SPI Resource Adapter. Follow the same steps to change the thread pool to a different one, if so desired. Note that even with a dedicated thread pool, all MDBs associated with the resource adapter still share the same thread pool. However, they do not have to compete with other WebSphere Application Server tasks that also use the default thread pool.

**Tune intermediate components for concurrency**
If the input business object is handled by a single thread from end to end, the tuning for the edge components is normally adequate. In many situations, however, there are multiple thread switches during the end to end execution path. It is important to tune the system to ensure adequate concurrency for each asynchronous segment of the execution path.

Asynchronous invocations of an SCA component utilize an MDB to listen for incoming events that arrive in the associated input queue. Each SCA module defines an MDB and its corresponding activation spec (JNDI name: sca/module name/ActivationSpec). Note that the SCA module MDB is shared by all asynchronous SCA components within the module, including SCA export components. Take this into account when configuring the activation specification’s maxConcurrency property value. SCA module MDBs use the same default thread pool as those for JMS exports.

The asynchrony in a long running business process occurs at transaction boundaries. (See 2.3, “Transactionality considerations” on page 16 for more details about settings that affect
transaction boundaries.) BPE defines an internal MDB and its activation specification, BPEInternalActivationSpec. You need to tune the `maxConcurrency` parameter following the same guidelines as for SCA module and JMS export MDBs. The only difference is that there is one BPEInternalActivationSpec in the WebSphere Process Server server.

**Configure JMS and JMS queue connection factories**

Multiple concurrently running threads can bottleneck on resources such as JMS and database connection pools if such resources are not tuned properly. The Maximum Connections pool size specifies the maximum number of physical connections that can be created in this pool. These are the physical connections to the back-end resource, for example a DB2 database. Once the thread pool limit is reached, no new physical connections can be created and the requester waits until a physical connection that is currently in use is returned 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. The threads waiting for connections to underlying resource are blocked until the connections are freed up and allocated to them by the pool manager. If no connection is freed in the specified interval, a `ConnectionWaitTimeout` exception is issued.

If the Maximum Connections property 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.

The general guideline for tuning connection factories is that their maximum connection pool size needs to match the number of concurrent threads multiplied by the number of simultaneous connections per thread.

For each import component with a JMS, MQ, or MQ JMS binding, there is a connection factory created during application deployment (for example SAPAdpt.JMSImport_CF in ContactManager). The Maximum Connections property of the JMS connection factory's connection pool should be large enough to provide connections for all threads concurrently executing in the import component. For example, if 100 threads are expected to run in a given module, the Maximum Connections property should be set to 100. The default is 10.

**Configure data source options**

The Maximum Connections property of data sources should be large enough to allow concurrent access to the databases from all threads. Typically there are a number of data sources configured in WebSphere Process Server or WebSphere ESB servers, for example BPEDB data source, WPSDB data source, and messaging engine database data sources. Set each data source's maximum connection property to match the maximum concurrency of other system resources.

**Set data source prepared statement cache size**

The BPC container uses prepared statements extensively. The statement cache sizes should be large enough to avoid repeatedly preparing statements for accessing the databases. The prepared statement cache for the BPEDB data source should be at least 128.
3.4.4 Messaging tuning

This section discusses messaging tuning.

For message engines, choose data store or file store

Message Engine persistence is usually backed by a database. For a stand-alone configuration of WebSphere Process Server or WebSphere ESB V6.1.0, the persistence storage of BPE and SCA buses can be backed by the file system (file store). The choice of file store has to be made at profile creation time. Use the Profile Management Tool to create a new stand-alone enterprise service bus profile or stand-alone process server profile. Choose Profile Creation Options → Advanced profile creation → Database Configuration, and select Use a file store for Messaging Engine (ME). When this profile is used, file stores are for BPE and SCA service integration buses.

Set data buffer sizes (discardable or cached)

The DiscardableDataBufferSize is the size in bytes of the data buffer used when processing best effort non persistent messages. The purpose of the discardable data buffer is to hold message data in memory, because this data is never written to the data store for this Quality of Service. Messages which are too large to fit into this buffer will be discarded.

The CachedDataBufferSize is the size in bytes of the data buffer used when processing all messages other than best effort non persistent messages. The purpose of the cached data buffer is to optimize performance by caching in memory data that might otherwise need to be read from the data store.

You can set the DiscardableDataBufferSize and CachedDataBufferSize under Service Integration → Buses → bus name → Messaging Engines → messaging engine name → Additional properties → Custom properties.

Move the messaging engine data stores to a high performance DBMS

For better performance, the message engine data stores should use production quality databases, such as DB2, rather than the default Derby. You can make this choice at profile creation time using the advanced profile creation option. If the profile was created already with Derby as the messaging engine data store, you can use the method that we describe here to change the data store to an alternative database.

After the Profile Management Tool has finished and Business Process Choreographer is configured, the system should contain four buses with one message engine each. Table 3-1 shows the buses in WebSphere Process Server installed on system box01. The node and cell names are the default names.

<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>

Each of these message engines is configured by default to use a data store in Derby. Each data store is located in its own database. For DB2, this configuration is not optimal from an administrative point of view. There are already many databases in the system, and adding
four more databases increases the maintenance and tuning effort substantially. The solution that we propose here uses a single DB2 database for all four data stores. The individual data stores (set of tables) are completely separate and each message engine acquires an exclusive lock on its set of tables during startup. Each message 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 Messaging engines and data stores in the WebSphere Application Server V6 information center at:

http://www.ibm.com/software/webservers/appserv/was/library/

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

Instead of having a DB2 database per messaging engine, we put all messaging engines into the same database using different schemas to separate them, as shown in Table 3-2.

<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>

Create one schema definition for each message engine with the Windows command shown in Example 3-1.

**Example 3-1 Command to create one schema definition**

```plaintext
<WPS Install>\bin\sibDDLGenerator.bat -system db2 -version 8.1 -platform windows -stmtend ; -schema BPCMSG -user <user> >createSIBSchema_BPCMSG.ddl
```

Repeat this command for each schema/messaging engine.

To distribute the database across several disks, edit the created schema definitions and put each table in a table space named after the schema used, that is SCAAPP becomes SCANODE_TS, CEIMSG becomes CEIMSG_TS and so on. The schema definition should look similar to Example 3-2 after editing.

**Example 3-2 Schema definition**

```sql
CREATE SCHEMA CEIMSG;
CREATE TABLE CEIMSG.SIBOWNER (
  ME_UUID VARCHAR(16),
  INC_UUID VARCHAR(16),
  VERSION INTEGER,
  MIGRATION_VERSION INTEGER
)IN CEIMSG_TB;
CREATE TABLE CEIMSG.SIBCLASSMAP (
  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 distribution depend on application structure and load characteristics. This example uses one table space per data store:

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

2. Create the table spaces and distribute the containers across available disks by issuing the following command for a system managed table space:

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

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

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

See 3.4.8, “Database: General tuning” on page 43 and 3.4.9, “Database: DB2 specific tuning” on page 44 for further information about database and DB2-specific tuning, respectively.

### Create the data sources for the messaging engines

Create a data source for each message engine and configure each message engine to use the new data store using the administrative console. Table 3-3 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</td>
<td>Derby JDBC Provider (XA)</td>
</tr>
<tr>
<td>box01-server1.SCA.APPLICATION.box01Node01Cell.Bus</td>
<td>Derby JDBC Provider</td>
</tr>
<tr>
<td>box01-server1.CommonEventInfrastructure_Bus</td>
<td>Derby JDBC Provider</td>
</tr>
<tr>
<td>box01-server1.BPC.box01Node01Cell.Bus</td>
<td>Derby JDBC Provider</td>
</tr>
</tbody>
</table>

Create a new JDBC provider, DB2 Universal JDBC Driver Provider, for the non-XA data sources first if it is missing. The XA DB2 JDBC Driver Provider should exist if BPC was configured correctly for DB2.

Create four new JDBC data sources, one for CEI as an XA data source, the remaining three as single-phase commit (non-XA) data sources. Table 3-4 provides new names.

<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 a data source:

1. Clear the “Use this Data Source in container managed persistence (CMP)” option.
2. Set a Component-managed authentication alias.
3. Set the database name to the name that is used for the database that was created earlier for messaging (for example SIB).
4. Select a driver type 2 or 4. Per DB2 recommendations, use the JDBC Universal Driver Type 2 connectivity to access local databases and Type 4 connectivity to access remote databases. Note that a driver of Type 4 requires a host name and valid port to be configured for the database.

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

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

1. In the Navigation Panel go to Service Integration → Buses, and change the data stores for each bus and messaging engine displayed.

2. Put in the new JNDI and schema name for each data store. Clear the Create Tables option because the tables are created already.

3. The server immediately restarts the message engine. The SystemOut.log shows the results of the change and also shows if the message engine starts successfully.

4. Restart the server and validate that all systems come up using the updated configuration.

The last remaining task is tuning the database. See 3.4.8, “Database: General tuning” on page 43 and 3.4.9, “Database: DB2 specific tuning” on page 44 for further information about database and DB2-specific tuning, respectively.

### 3.4.5 WebSphere ESB tuning

In this section, we discuss additional configuration options that are relevant to tuning WebSphere ESB. See 4.2, “WebSphere Enterprise Service Bus settings” on page 59 for a description of specific WebSphere ESB parameter settings.

**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 might also find benefit in tuning the connection pooling and statement cache of the data source. See 3.4.8, “Database: General tuning” on page 43 and 3.4.9, “Database: DB2 specific tuning” on page 44 for further information about tuning DB2.

**Disable event distribution for CEI**

The Event Server which manages events can be configured to distribute events or log them to the event database. Some mediations only require events to be logged to a database. For these cases, performance is improved by disabling event distribution. Because the event server can be used by other applications it is important to check that none of them use event monitoring which requires event distribution before disabling this.

You can disable event distribution from Resources → Common Event Infrastructure → Cell scope → Event Server Profile. Then, clear EnableEventDistribution.

**Configure WebSphere Service Registry and Repository cache timeout**

WebSphere Service Registry and Repository (WSRR) is used by WebSphere ESB for endpoint lookup. When accessing the WebSphere Service Registry and Repository (for example, using the endpoint lookup mediation primitive), results from the registry are cached in WebSphere ESB. You can configure the lifetime of the cached entries using Service Integration → WSRR Definitions → WSRR definition name → Timeout of Cache.

Validate that the timeout is a sufficiently large value, the default timeout in V6.1.1 is 300 seconds, which is reasonable from a performance perspective. Too low a value results in frequent lookups to the WebSphere Service Registry and Repository, which can be expensive.
(especially if retrieving a list of results) and also includes the associated network latency if the registry is located on a remote machine.

**Configure WebSphere ESB resources**

When creating resources using the WebSphere Integration Developer Tooling, the application developer is given the choice to use pre-configured WebSphere ESB resources or to let the Tooling generate the Mediation Flow related resources that it requires. Both approaches have their advantages and disadvantages.

Preconfigured resources support:
- Existing resources to be used
- External creation and tuning scripts to be applied
- Easier post deployment adjustment

Tooling created resources support:
- No further need for creating resources using scripts or the administrative console
- The ability to change the majority of performance tuning options as they are now exposed in the Tooling

In our performance tests, we used preconfigured resources because by segregating the performance tuning from the Business logic, the configuration for different scenarios can be maintained in a single script. It is also easier to adjust these parameters after the applications are deployed.

The only cases where this pattern has not been followed is for Generic JMS bindings. In these scenarios where resources have already been configured by the 3rd party JMS provider software (MQ 6.0.2.2 for all instances in this report), the Tooling created resources are used to locate the externally defined resources.

### 3.4.6 WebSphere Adapters tuning

This section discusses configuration options that are relevant to tuning applications that use WebSphere Adapters.

**Configure the poll period and poll quantity**

Two of the most important configuration parameters for the WebSphere Adapters are:
- `pollPeriod` specifies the amount of time (in milliseconds) between polling actions.
- `pollQuantity` specifies the maximum number of events to process during a polling action.

You can modify these settings in the activation specification of the adapter application. These settings affect only the performance of polling (inbound) applications.

Because these parameters control the rate and amount of work that an adapter processes, the combination of `pollPeriod` and `pollQuantity` regulate the number of transactions that are processed first by the adapter, and then by the broker (for example 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 either low system throughput (if `pollPeriod` is too long or `pollQuantity` is too low), or 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 `pollPeriod` is too short or `pollQuantity` is too high). Because both of these conditions impact overall system performance dramatically, appropriate settings for
**pollPeriod** and **pollQuantity** are critical and should be configured explicitly to support the level of throughput a solution is designed to handle.

There are two possible scenarios that have to be kept 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, the recommendation is to configure **pollPeriod** and **pollQuantity** to enable events to be retrieved and processed at a level that matches the peak throughput of the overall solution. This is discussed in more detail below.

As an 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 **pollPeriod** to some small value (perhaps 10 milliseconds) and set **pollQuantity** 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 **pollPeriod** setting enables a minimal delay between poll cycles.

Factors that can require an adjustment to these values include:

- **The size of the object that is processed**
  - For larger objects, a good rule of thumb is to use a lower **pollQuantity** and longer **pollPeriod**. This does not generally apply for 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), in order to avoid potential OutOfMemory exceptions. To extend the above example, if the object size is 1 MB and the throughput rate of the solution is 2 events per second, appropriate settings could be **pollPeriod** = 100 milliseconds and **pollQuantity** = 2. Also note that the poll period can be more finely adjusted by actually calculating the time it takes to process the **pollQuantity** events (2 in this case) and adjusting the poll period for that.

- **The Java heap size and physical memory that is available on the system**
  - In general, the larger the heap, the higher **pollQuantity** can be set. However, there are several factors involved in setting the heap size, one very important factor being to ensure that the heap is not set so large that paging results. Paging the Java heap can reduce system performance dramatically. See 3.4.11, “Advanced Java heap tuning” on page 48 for a detailed discussion on setting the Java heap sizes appropriately. Note that the heap requirement for an object is really a multiple of its size. A good rule of the thumb is that three times the size of the object but that size can vary depending on the adapter and the business process.

- **The uniformity of event arrival rates**
  - The examples that we include here assume that events arrive at a relatively constant rate, which might not be true for many solutions. Event arrival is sometimes very uneven. In these cases, you must take care to balance processing events in a timely manner in order to handle the occasional high arrival rates, while also not holding too many events in memory concurrently and potentially encountering OutOfMemory exceptions.

- **Another consideration when processing very large objects (large files for example)** is the number of objects in-flight at any given time. When using the Flat Files adapter it is best to reduce the amount of large files in-flight at a given time. If a large file is being processed using Split Criteria, the adapter will start processing **pollQuantity** number of events from the file. The adapter then starts preparing other files in the event delivery directory for processing before the first file has completed, possibly leading to an OutOfMemory
situation. Depending on the file size and processing rate, it might be advisable to limit the number of files in the delivery directory to avoid this issue. If too many files are in-flight at a given time, then memory is constrained and either OutOfMemory exceptions can occur or garbage collection time is increased, which results in reduced performance.

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. Set the number of threads to a value that includes the needs of the server and applications plus the JCA adapter. To set the number of threads using the administrative console:

1. Select **Servers → Application servers → server name → Additional Properties → Thread Pools → Default**.
2. Then, update the Maximum and Minimum values.
3. Select OK and save.

These settings, along with **pollQuantity** and **pollPeriod**, allow more objects to be in-flight at a single time (for polling adapters). So, 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.

Configure the work manager thread pool
For adapters performing Request Processing that use their own thread pool or use the default work manager, the number of threads in the pool will need to be tuned.

To set this using the administrative console, select **Resources → Asynchronous beans → Work Managers → DefaultWorkManager** (or your application-specific work manager). Then, set the Minimum and Maximum number of threads.

We suggest 20 for both minimum and maximum as a starting point, but depending on your configuration, you might need higher values.

Flat Files and FTP: Place event delivery and archive directories on the same disk
Placing the delivery and archive directories on the same physical disks enables an optimization during the archiving process, which uses a fast rename operation in this case instead of a slower (especially for large files) copy of the file from the event delivery directory to the archive directory.

Monitor JMS queues and external event table for stale events
Certain adapters such as Flat Files, FTP, e-mail, and SAP® can use external event tables. JMS queues can also be used at various points, for example during asynchronous event delivery. You should periodically monitor both the tables and JMS queues to ensure events are being processed and removed from these queues. Obsolete events can produce delays in processing, because the adapter might attempt to unnecessarily recover these events. External table names can be found by using the administrative console. The names are specified in the J2C Activation Spec of the deployed adapter. These tables can then be monitored, and cleared, using the appropriate database administration tool (for example, if DB2 is your database use the DB2 Control Center). The relevant JMS queues can also be found using the administrative console. Again, if using DB2, they are specified as data sources in the Service Integration and Resources section.
You can monitor the relevant tables and queues 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 start the application.

### 3.4.7 WebSphere Business Monitor tuning

Figure 3-1 shows an example of a WebSphere Business Monitor installation. An event emitter, such as business processes running on WebSphere Process Server, delivers events to CEI, which then delivers them to the JMS queues that are associated with the monitor models that have registered to receive such events. Each monitor model receives events in a dedicated queue. WebSphere Business Monitor Server takes events from these queues, code specific to each model processes them, and relevant information is added to the WebSphere Business Monitor database. WebSphere Business Monitor dashboards can then be used to offer various views of the data through a Web browser. The WebSphere Business Monitor Dashboard applications translate user activities at the dashboard into SQL or MDX queries directed at the WebSphere Business Monitor database.

Optionally, you can enable WebSphere Business Monitor Data Movement Service (DMS) to copy information periodically from the event processing tables to the dashboard tables and then prune the copied data from the event processing tables. This feature separates event processing database access from dashboard queries and so helps to provide better system responsiveness.

![Figure 3-1 WebSphere Business Monitor 6.1 configuration](image)

**Configure Java heap sizes**

The default initial and maximum heap sizes in most implementations of Java are too small for many of the servers in this configuration. The WebSphere Business Monitor Launchpad...
installs WebSphere Business Monitor and its prerequisite servers with larger heap sizes, but you might check that these sizes are appropriate for your hardware and workload. We used initial and maximum heap sizes of 1536 MB and 1536 MB, respectively, on all our WebSphere servers when producing the results reported in this document. Also see 4.3, “WebSphere Business Monitor settings” on page 61 for more details.

Configure CEI
By default, when an event arrives at CEI, it is put into the queue you have specified (in this case into a queue for a particular WebSphere Business Monitor Model) and into an additional, default queue. Unless the customer needs this extra queue, do not double-store. You can prevent this extra store in the administrative console by removing the All Events event group found using Service Integration → Common Event Infrastructure → Event Service → Event Services → Default Common Event Infrastructure event server → Event Groups.

The JMS queue from which WebSphere Business Monitor takes events is persistent, and by default the WebSphere default messaging provider uses the Derby database management system for its persistent store. We have found that performance can be improved by replacing Derby 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 from the administrative console using Service Integration → Common Event Infrastructure → Event Service → Event Services → Default Common Event Infrastructure event server. Then, clear Enable Data Store.

If you have CEI and WebSphere Business Monitor Server on separate platforms, as shown in Figure 3-1, then we recommend that the WebSphere Business Monitor model queues reside physically on a messaging bus that is co-located with the WebSphere Business Monitor Server. The Distribution Queues specified in the CEI Event Groups for the installed WebSphere Business Monitor Models should then be set up to refer to these queues by providing the remote queue’s messaging bus and connection endpoint information in the queue connection factory.

We also set the custom properties as shown in Table 3-5 for the messaging engine that is used to deliver messages to the WebSphere Business Monitor model queues.

**Table 3-5  Set custom properties for messaging engine**

<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 set these properties using the administrative console by selecting Service integration → buses → Monitor bus name → Messaging engines → engine name → Custom properties.

Set Message Consumption Batch Size
Taking events off the CEI queue in large batches is much more efficient than one at a time. Up to some limit, the larger the batch size, the higher event processing throughput will be. However, there is a trade-off. Taking some events off the CEI queue, processing them, and persisting them to the WebSphere Business Monitor database is done as a transaction. So, although a larger batch size yields better throughput, it can cost more if you have to roll back. By default, WebSphere Business Monitor sets the batch size to 100 events.
If you experience frequent rollbacks, consider reducing the batch size in the administrative console using the server scope as follows:

- Applications → Monitor Models → version → RuntimeConfiguration → General → Message Consumption Batch size: <default 100>

WebSphere Business Monitor processes events most efficiently when the Deserialization WorkManager work request queue is the same size as the Message consumption Batch size. You can set the size of the work request queue through the administrative console using Resources → Asynchronous beans → Work managers → wbm_ model _version_Deserializer:

- Work request queue size: <same as batch size>
- Also, make sure that growable is not selected.

Deserialization and FragmentProcessing WorkManager maximum threads

The processing of incoming events for a model is done through the use of several WorkManager thread pools. You can change the maximum number of threads for two thread pools to get full use of all available CPUs on a WebSphere Business Monitor Server:

- Deserialization
- FragmentProcessing

For maximum throughput, a good starting point for the Deserialization WorkManager's maximum number of threads is one thread per available processor core. The corresponding setting for the FragmentProcessing WorkManager is twice the number of available processor cores.

Of course, the number of installed models and their event processing requirements might require adjustments to this simple rule. In general, models which have lower throughput requirements should probably have smaller thread pools.

3.4.8 Database: General tuning

This section discusses general database tuning.

Place database log files on a fast disk subsystem

Databases are designed for high availability, transactional processing and recoverability. Because for performance reasons changes to table data cannot be written immediately to disk, these changes are made recoverable by writing to the database log. Updates are made to database log files when log buffers fill and at transaction commit time. As a result, database log files can be utilized heavily. More importantly, log writes often hold commit operations pending, meaning that the application is synchronously waiting for the write to complete. Therefore, 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.

Place logs on separate device from table space containers

A basic strategy for all database storage configurations is to place the database logs on separate physical disk devices from the table space containers. This reduces disk access contention between IO to the table space containers and IO to the database logs and preserves the mostly sequential access pattern for the log stream. Such separation also improves recoverability when log archival is employed.
3.4.9 Database: DB2 specific tuning

Providing a comprehensive DB2 tuning guide is beyond the scope of this report. However, there are a few general rules of thumb that can assist in improving the performance of DB2 environments. In this section, we discuss these rules and provide pointers to more detailed information. You can find a quick reference for DB2 performance tuning in the developerWorks® article DB2 Tuning Tips for OLTP Applications, which is available at:

You can find the complete set of current DB2 manuals (including database tuning guidelines) in the DB2 Information Center by searching for download pdf:
http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp

We highly recommend that you consult the document entitled Administration Guide: Performance.

The DB2 Information Center also contains a wealth of performance related information and is easy to search.

Update database statistics
It is important to update the statistics so that DB2 can optimize accesses to key tables. 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. See Improving the performance of complex BPC API queries on DB2, which is available at:
http://www-1.ibm.com/support/docview.wss?rs=2307&uid=swg21299450

DB2 8.2 contains new 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. You can turn on this feature from the DB2 Control Center. Updating statistics allows the DB2 query optimizer to create better performing access plans for evaluating queries.

One approach to updating statistics manually on all tables in the database is using the REORGCHK command. Dynamic SQL, such as that produced by JDBC, is taken immediately to the new statistics into account. Static SQL, such as that in stored procedures, must be rebound explicitly in the context of the new statistics. Run these commands on database DBNAME:
db2 connect to DBNAME
db2 reorgchk update statistics on table all
db2 connect reset
db2rbind DBNAME all

Execute the REORGCHK and rebind when the system is relatively idle so that you can acquire a stable sample and can avoid possible deadlocks in the catalog tables.

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 can 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, DB2 recommends that the number of pages used by a buffer pool be set explicitly. This can be done using either the CREATE BUFFERPOOL or the ALTER BUFFERPOOL commands. The default number of pages is 250, resulting in a quite small total default buffer pool size of 4 KB * 250 = 1000 KB.

**Note:** All buffer pools reside in database global memory, which is allocated on the database system. The buffer pools must coexist with other data structures and applications, all without exhausting available memory. In general, having larger buffer pools can improve 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. Be careful to avoid configuring large buffer pool size settings, which lead to paging activity on the system.

### Maintain proper table indexing

While the WebSphere BPM products create a set of database indexes that are appropriate for many installations, additional indexes might be required in some circumstances. A database environment that requires additional indexes will often exhibit performance degradation over time; in some cases the performance degradation can be profound. Environments that need additional indexes often exhibit heavy read IO on devices holding the table space containers. To assist in determining which additional indexes could improve performance, DB2 provides the Design Advisor. The Design Advisor is available from the DB2 Control Center, or it can be started from a command line processor. It has the capability to help define and design indexes suitable for a particular workload.

### Size log files appropriately

When using circular logging, it is important that the available log space permits dirty pages in the bufferpool to be cleaned at a reasonably low rate. Changes to the database are immediately written to the log, but a well tuned database will coalesce multiple changes to a page before eventually writing that modified page back to disk. Naturally, changes recorded only in the log cannot be overwritten by circular logging. DB2 detects this condition and forces the immediate cleaning of dirty pages required to allow switching to a new log file. Although this mechanism protects the changes recorded in the log, all application logging must be suspended until the necessary pages are cleaned.

DB2 works to avoid pauses when switching log files by proactively triggering page cleaning under control of the database level `softmax` parameter. The default value of 100 for `softmax` begins background cleaning activities when the gap between the current head of the log and the oldest log entry recording a change to a dirty page exceeds 100% of one log file in size. In extreme cases this asynchronous page cleaning cannot keep up with log activity, leading to log switch pauses which degrade performance.

Increasing the available log space gives asynchronous page cleaning more time to write dirty bufferpool pages and avoid log switch pauses. A longer interval between cleanings allows multiple changes to be coalesced on a page before it is written, which reduces the required write throughput by making page cleaning more efficient.

Available log space is governed by the product of log file size and the number of primary log files that are configured at the database level. The `logfilsiz` setting is the number of 4 KB pages in each log file. The `logprimary` setting controls the number of primary log files. As a starting point, try using 10 primary log files which are large enough that they do not wrap for at least a minute in normal operation.
Increasing the primary log file size does have implications for database recovery. Assuming a constant value for \( \text{softmax} \), larger log files mean that recovery can take more time. The \( \text{softmax} \) parameter can be lowered to counter this, but keep in mind that more aggressive page cleaning can also be less efficient. Increasing \( \text{softmax} \) gives additional opportunities for write coalescing at the cost of longer recovery time.

**Ensure that sufficient locking resources are available**

Locks are allocated from a common pool controlled by the database level parameter `locklist`, which is the number of 4 KB pages that are set aside for this use. A second database level parameter, `maxlocks`, limits the percentage of the lock pool that is held by a single application. When an application attempts to allocate a lock that exceeds the fraction allowed by `maxlocks` or when the free lock pool is exhausted, DB2 performs lock escalation to replenish the supply of available locks. Lock escalation involves replacing many row locks with a single table-level lock.

While lock escalation addresses the immediate problem of lock pool overuse or starvation, it can lead to database deadlocks, and so should not occur frequently during normal operation. In some cases, you can alter application behavior to reduce pressure on the lock pool by breaking up large transactions which lock many rows into smaller transactions. It is usually simpler to try tuning the database first.

Beginning with Version 9, DB2 adjusts the `locklist` and `maxlocks` parameters automatically by default. To manually tune these parameters, first observe whether lock escalations are occurring either by examining `db2diag.log` or by using the system monitor to gather snapshots at the database level. If the initial symptom is database deadlocks, also consider whether these are initiated by lock escalations.

- Check the Lock escalations count in the output from:
  ```
  db2 get snapshot for database yourDatabaseName
  ```

- Obtain the current values for `locklist` and `maxlocks` by examining the output from:
  ```
  db2 get db config for yourDatabaseName
  ```

- Alter these values, for example to 100 and 20, as follows:
  ```
  db2 update db config for yourDatabaseName using locklist 100 maxlocks 20
  ```

When increasing the `locklist` size, consider the impacts of the additional memory allocation required. Often the `locklist` size is relatively small compared with memory dedicated to buffer pools, but the total memory required must not lead to virtual memory paging.

When increasing the `maxlocks` fraction, consider whether a larger value will allow a few applications to drain the free lock pool, leading to a new cause of escalations as other applications needing relatively few locks encounter a depleted free lock pool. Often it is better to start by increasing the `locklist` size alone.
3.4.10 Database: Oracle specific tuning

As with DB2, providing a comprehensive Oracle database tuning guide is beyond the scope of this paper. However, there are a few general rules of thumb that can assist in improving the performance of Oracle environments. In this section, we discuss these rules and provide pointers to more detailed information. In addition, the following references are useful:

- Oracle Database 10g Release 2 documentation (includes a Performance Tuning Guide), available at:  
  http://www.oracle.com/technology/documentation/database10gr2.html
- Introduction to Oracle database performance tracing, available at:  
  http://www.oracle.com/technology/oramag/oracle/04-jan/o14tech_perf.html

Update database statistics

It is important to update the statistics so that Oracle can optimize accesses to key tables. Statistics are used by the Oracle cost based 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. Updating statistics allows the query optimizer to create better performing access plans for evaluating queries.

One approach to updating statistics manually on all tables in a schema is to use the dbms_stats utility as shown in Example 3-3.

```
Example 3-3   Using dbms_stats utility

execute dbms_stats.gather_schema_stats( -
    ownname => 'your_schema_name', -
    options => 'GATHER AUTO', -
    estimate_percent => DBMS_STATS.AUTO_SAMPLE_SIZE, -
    cascade => TRUE, -
    method_opt => 'FOR ALL COLUMNS SIZE AUTO', -
    degree => 15);
```

Set buffer cache sizes correctly

The following reference discusses this issue in detail:

http://download-uk.oracle.com/docs/cd/B19306_01/server.102/b14211/memory.htm#i29118

Size log files appropriately

Unlike DB2, Oracle performs an expensive checkpoint operation when switching logs. The checkpoint involves writing all dirty pages in the buffer cache to disk. Therefore, it is important to make the log files large enough that switching occurs infrequently. Applications that generate a high volume of log traffic need larger log files to achieve this goal.

Maintain proper table indexing

While the WebSphere BPM products create a set of database indexes that are appropriate for many installations, additional indexes might be required in some circumstances. A database environment that requires additional indexes will often exhibit performance degradation over time; in some cases the performance degradation can be profound. Environments that need additional indexes often exhibit heavy read IO on devices holding the table space data files. To assist in determining which additional indexes could improve performance, Oracle 10g provides the Automatic Database Diagnostic Monitor. It has the capability to help define and design indexes suitable for a particular workload.
3.4.11 Advanced Java heap tuning

Because the WebSphere BPM 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 can be used to improve both tooling and runtime performance. The most important of these are related to garbage collection and setting the Java heap size. This section discusses these topics in detail.

The products covered in this report utilize IBM JVMs on most platforms (AIX, Linux®, Windows, and so forth), and the HotSpot JVMs on selected other systems, such as Solaris™. Vendor specific JVM implementation details and settings will be discussed as appropriate. Also note that all V6.1.0 products in this document use Java 5, which is much different from Java 1.4.2 used by V6.0.2.x and earlier releases. For brevity, we discuss only Java 5 tunings here.

To view the IBM Java 5 Diagnostics Guide, go to:
http://publib.boulder.ibm.com/infocenter/javasdk/v5r0/index.jsp

This guide discusses many more tuning parameters than those that we discuss in this paper, but most parameters are for specific situations and are not of general use. For a more detailed description of IBM Java 5 garbage collection algorithms, see the section “Garbage Collectors” in the chapter “Understanding the IBM JDK™ for Java” in the diagnostics guide.

Sun™ HotSpot JVM references include:
- For a useful summary of HotSpot JVM options for Solaris, go to:
  http://java.sun.com/docs/hotspot/VMOptions.html
- For a useful FAQ about the Sun HotSpot JVM, go to:
  http://java.sun.com/docs/hotspot/PerformanceFAQ.html#20
- For more performance tuning information of Sun's HotSpot JVM, go to:
  http://java.sun.com/docs/performance/

Monitoring garbage collection

To set the heap correctly, you must first determine how the heap is being used by collecting a verbosegc trace. A verbosegc trace prints garbage collection actions and statistics to stderr in IBM JVMs and stdout in Sun HotSpot JVMs. The verbosegc trace is activated using the Java run-time option verbosegc. Output from verbosegc is different for the HotSpot and IBM JVMs, as shown in Example 3-4.

Example 3-4 IBM JVM verbosegc trace output

```
<af type="tenured" id="12" timestamp="Fri Jan 18 15:46:15 2008"
 intervalms="86.539">
  <minimum requested_bytes="3498704" />
  <time exclusiveaccessms="0.103" />
  <tenured freebytes="80200400" totalbytes="268435456" percent="29" >
    <soa freebytes="76787560" totalbytes="255013888" percent="30" />
    <loa freebytes="3412840" totalbytes="13421568" percent="25" />
  </tenured>
</af>
<gc type="global" id="12" totalid="12" intervalms="87.124">
  <refs_cleared soft="2" threshold="32" weak="0" phantom="0" />
  <finalization objectsqueued="0" />
  <timesms mark="242.029" sweep="14.348" compact="0.000" total="256.598" />
  <tenured freebytes="95436688" totalbytes="268435456" percent="35" >
```
Example 3-5 shows Sun HotSpot JVM verbosegc trace output (young and old).

**Example 3-5  Sun HotSpot JVM verbosegc trace output**

```
[GC 325816K->83372K(776768K), 0.2454258 secs]
[Full GC 267628K->83769K <- live data (776768K), 1.8479984 secs]
```

Sun HotSpot JVM verbosegc output can be more detailed by setting additional options:

-XX:+PrintGCDetails -XX:+PrintGCTimeStamps

It is tedious to parse the verbosegc output using a text editor. There are very good visualization tools on the Web that can be used for more effective Java heap analysis. The IBM Pattern Modeling and Analysis Tool (PMAT) for Java Garbage Collector is one such tool. IBM Pattern Modeling and Analysis Tool for Java Garbage Collector is available for free download at IBM alphaWorks:


PMAT supports the verbosegc output format of JVMs offered by major JVM vendors such as IBM, Sun, and HP.

**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 50. If your objective is to support large business objects, read 3.4.2, “Tuning for large objects” on page 30.

For most production applications, the IBM JVM Java heap size defaults are too small and should be increased. In general the Sun HotSpot JVM default heap and nursery size are also too small and should be increased (we will show how to set these parameters later).

There are several approaches to setting optimal heap sizes. We describe here the approach that most applications should use when running the IBM JVM on AIX. The essentials can be applied to other systems. 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 will then try to keep the GC time within reasonable limits by growing and shrinking the heap. You then use the output from verbosegc to monitor GC activity.

If Generational Concurrent GC is used (-Xgcpolicy:gencon), the new area size can also be set to specific values. By default, the new size is a quarter of the total heap size or 64 MB,
whichever is smaller. For better performance, the nursery size should be ¼ of the heap size or larger, and it should not be capped at 64MB. New area sizes are set by JVM options: 
-Xmn<size>, -Xms<initialSize>, and -Xmx<maxSize>

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 also increase the nursery size to approximately 1/4 of the heap size. Note that you should never increase the nursery to more than 1/2 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, then use verbosegc traces to monitor GC activity. After analyzing the output, 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, throughput can be improved 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. Note that increasing the maximum size of the Java heap might not always solve this type of problem because it could be a memory over-usage problem. Conversely, if response times are too long due to GC pause times, 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 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 will be 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 and stabilizes at that point. Based on this, set the initial heap size to 256 MB (-Xms256m).
- Based on the verbosegc trace output, set the maximum heap size appropriately. Care must be taken to not set this value too low, or OutOfMemory errors will occur. The maximum heap size must be large enough to allow for peak throughput. Using the example in the previous bullet item, a maximum heap size of 768 MB might be appropriate (-Xmx768m). This is to give the Java heap “head room” to expand beyond its current size of 400 MB if required. Note that the Java heap will only grow if required (e.g., if a period of peak activity drives a higher throughput rate), so setting the maximum heap size somewhat higher than current requirements is generally a good policy.
- Be careful to not set the heap sizes too low, or garbage collections will occur frequently, which might reduce throughput. Again, a verbosegc trace can assist 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.

**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 “Monitoring garbage collection” on page 48 for further information about `verbosegc` output. Many garbage collections that produce very little free heap space generally occurs preceding this exception. If this is the problem, then increase the size of the heap.

If, however, 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 provides this information). If these appear high, then a subtle effect might be occurring whereby resources outside the heap are held by objects within the heap and being cleaned by finalizers. Reducing the size of the heap can alleviate this situation, by increasing the frequency with which finalizers are run.

**Note:** 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 can force a thread stack overflow in case of insufficient stack size.

For middleware products, if you are using an in-process version of the JDBC 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’s Net drivers and Oracle’s Thin drivers) and you can switch MQSeries® from Bindings mode to Client mode and so on. Refer to the documentation for the products in question for more details.

**Set AIX threading parameters**

The IBM JVM threading and synchronization components are based upon the AIX POSIX compliant Pthreads implementation. The environment variables shown in Example 3-6 are found to improve Java performance in many situations. We used them for the workloads in this paper. The variables control the mapping of Java threads to AIX Native threads, turn off mapping information, and allow for spinning on mutually exclusive (mutex) locks.

```
Example 3-6   Environment variables

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

You can find more information about AIX specific Java tuning at:

3.4.12 i5/OS tuning

This section discusses tuning and configuration recommendations that are specific to i5/OS®. Note that, unless indicated otherwise, the tuning options listed above also apply to i5/OS.

The tuning categories are:

- OS specific
- Java Virtual Machine
- DB2

**OS specific tuning**

System i™ i5/OS has a unique memory management concept known as memory pools for monitoring memory usage. There are four memory pools defined by default:

- **MACHINE**: The main memory for System i jobs.
- **BASE**: The memory pool which contains all unassigned main storage on the system (all main storage that is not required by another memory pool). By default, WebSphere Process Server, WebSphere ESB, and DB2 run in this pool.
- **INTERACT**: The memory pool used for interactive jobs.
- **SPOOL**: The memory pool used for spool (print) writers.

It is recommended that you run Java applications (such as WebSphere Process Server and WebSphere ESB) in a separate memory pool. On systems running multiple workloads simultaneously, putting Java applications in their own pool will ensure that the Java applications have enough memory allocated to them. A common practice is to create one or more shared memory pools for WebSphere workloads running on System i. This is to ensure an adequate amount of memory is always available specifically for WebSphere. This is well described in *WebSphere Application Server for i5/OS Handbook: Version 6.1*, SG24-7221.

**Java Virtual Machine tuning**

i5/OS supports two JVMs:

- The IBM Technology for Java
- The Classic JVM (see Table 3-6 on page 53 for release dependent JVM availability)

The default JVM on i5/OS V5R4 release and earlier is Classic. However, the IBM Technology for Java generally performs better. To enable the WebSphere Process Server or WebSphere ESB profile installed to utilize a different JVM, execute the `enablejvm` command from the product's home directory. For example to enable the J9-32bit JVM for WebSphere Process Server:

```bash
cd /QIBM/ProdData/WebSphere/ProcServer/bin
enablejvm -jvm std32 -profile profilename
```

The IBM Technology for Java 32-bit JVM is limited to 4 GB of memory, allowing for a heap setting of 2.5 to 3.25 GB for most applications. If the application requires a large heap and you are using i5/OS V6R1, you can utilize the 64-bit IBM Technology for Java. In this case, substitute std32 with std64. Otherwise, consider utilizing the Classic 64-bit JVM and substitute std32 with classic.

The default initial and maximum heap settings on the Classic JVM are 96 MB and *NOMAX respectively. For WebSphere Process Server and WebSphere ESB workloads, it is recommended to increase the initial heap setting. See “Setting the heap size for most configurations” on page 49 to set the heap size for most configurations.
The Classic JVM yields even better performance when using the just in time compiler which is set by the \texttt{-Djava.compiler=jitc} property.

Further JVM tuning, JVM performance tools and recommendations are described in \textit{System i Performance Capabilities Reference i5/OS Version 5, Release 4 and IBM Java Diagnostic Guide 5.0}.

Table 3-6 shows the supported and default JVMs by OS level and Licensed Program Product (LPP).

\textbf{Table 3-6} \hspace{1cm} JVM support by operating system

<table>
<thead>
<tr>
<th>OS level</th>
<th>JVM</th>
<th>LPP</th>
<th>Default JVM</th>
<th>Java_Home</th>
</tr>
</thead>
<tbody>
<tr>
<td>V5R3</td>
<td>Classic-64 bit</td>
<td>Classic</td>
<td>/QIBM/ProdData/Java400/jdk*</td>
<td></td>
</tr>
<tr>
<td>V5R4</td>
<td>Classic-64 bit</td>
<td>5722JV1</td>
<td>Classic</td>
<td>/QIBM/ProdData/Java400/jdk*</td>
</tr>
<tr>
<td>V5R4</td>
<td>J9-32bits</td>
<td>5722JV1</td>
<td>J9-32bit</td>
<td>/QOpenSys/QIBM/ProdData/JavaVM/jdk*/32bit</td>
</tr>
<tr>
<td>V6R1</td>
<td>Classic</td>
<td>5761JV1</td>
<td>J9-32bit</td>
<td>/QIBM/ProdData/Java400/jdk*</td>
</tr>
<tr>
<td>V6R1</td>
<td>J9-32bits</td>
<td>5761JV1</td>
<td></td>
<td>/QOpenSys/QIBM/ProdData/JavaVM/jdk*/32bit</td>
</tr>
<tr>
<td>V6R1</td>
<td>J9-64bits</td>
<td>5761JV1</td>
<td></td>
<td>/QOpenSys/QIBM/ProdData/JavaVM/jdk*/64bit</td>
</tr>
</tbody>
</table>

\textbf{Note:} For the Classic JVM, also set the Java JDK property \texttt{java.version=1.4 (5.0 or 6.0)}

\textbf{DB2 tuning}

On i5/OS, there are two DB2 drivers that you can use for WebSphere workloads:

\begin{itemize}
  \item Native JDBC
  \item Toolbox
\end{itemize}

Native JDBC jobs run in the QSYSWRK subsystem with the job name QSQSRVR. Toolbox jobs run in QUSRWRK subsystem with the job name QZDASOINIT. The recommendation is to use Native JDBC if your database is local and Toolbox for remote database access.

Depending on the DB2 driver in use, a certain number of jobs are prestarted, which you might need to adjust. You can determine the number of prestarted jobs by running:

\begin{itemize}
  \item QSYS/DSPACTPJOB SBSD(QSYSWRK) PGM(QSQSRVR) {for the native driver}
  \item QSYS/DSPACTPJOB SBSD(QUSRWRK) PGM(QZDASOINIT) {for the toolbox driver}
\end{itemize}

After you have determined a suitable number of active database jobs, you can use the i5/OS Change Prestart Job feature to start these jobs at system startup time, as opposed to run time. For example:

\begin{itemize}
  \item CHGPJEB SBSD(QSYSWRK) PGM(QSQSRVR) INLJOBS(20) {set the initial number of jobs to 20}
  \item CHGPJEB SBSD(QUSRWRK) PGM(QZDASOINIT) INLJOBS(10) {set the initial number of jobs to 10}
\end{itemize}

To ensure that the database indexes are well tuned, consider using the System i Navigator to analyze and create indexes. The System i Navigator also shows the most frequently used SQL statements, and those that take the longest to execute.
In this chapter, we provide suggested initial settings for several relevant parameters. These values are not 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 proof-of-concepts and customer deployments. As discussed in 3.1, “Performance tuning methodology” on page 24, tuning is an iterative process. Follow that procedure, and adjust these values as appropriate for your environment.
4.1 WebSphere Process Server settings

The section describes settings that are used for selected internal performance evaluations of WebSphere Process Server. These settings were derived using the tuning methodology and guidelines described in Chapter 3, “Performance tuning and configuration” on page 23. Consider these settings useful starting points for your use of this product. For settings that we do not list, use the default settings that are supplied by the product installer as a starting point, and then follow the tuning methodology specified in 3.1, “Performance tuning methodology” on page 24.

We discuss two settings in this section:
- A two-tier (client/server) setup with the BPE database co-located on the server
- A three-tier setup with the BPE database located on a separate server

4.1.1 Two-tier configuration using JMS file store

This configuration was used in our internal performance work to evaluate the performance of a long-running business process that models a typical mortgage application process. The JMS binding is used for communication.

In this configuration, the BPE uses a DB2 database, and the messaging engines are configured to use file stores. To select the file store option, start the Profile Management Tool, select Advanced Profile Creation, and then on the Database Configuration window, select Use a file store for Messaging Engines (MEs).

Tuning parameter settings for the BPE database were initially derived using the DB2 Configuration Advisor. A few key parameter settings are modified further:
- MAXAPPLS, which must be large enough to accommodate connections from all possible JDBC Connection Pool threads
- The default buffer pool sizes (number of 4 KB pages in IBMDEFAULTBP) for each database are set so that each pool is 256 MB in size

Table 4-1 shows the parameter settings used for this evaluation.

<table>
<thead>
<tr>
<th>Parameter names</th>
<th>BPC DB settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>APP_CTL_HEAP_SZ</td>
<td>144</td>
</tr>
<tr>
<td>APPGROUP_MEM_SZ</td>
<td>13001</td>
</tr>
<tr>
<td>CATALOGCACHE_SZ</td>
<td>521</td>
</tr>
<tr>
<td>CHNGPGS_THRES</td>
<td>55</td>
</tr>
<tr>
<td>DBHEAP</td>
<td>600</td>
</tr>
<tr>
<td>LOCKLIST</td>
<td>500</td>
</tr>
<tr>
<td>LOCKTIMEOUT</td>
<td>30</td>
</tr>
<tr>
<td>LOGBUFSZ</td>
<td>245</td>
</tr>
<tr>
<td>LOGFILSZ</td>
<td>1024</td>
</tr>
<tr>
<td>LOGPRIMARY</td>
<td>11</td>
</tr>
</tbody>
</table>
In addition to these database level parameter settings, several other parameters are also modified using the administrative console, mostly those affecting concurrency (thread settings):

- The amount of expected concurrency influences the size of the thread pool, because more in-flight transactions requires more threads. For example, the size of the default thread pool is increased to a maximum of 30 threads on the Windows platform and to 50 threads on the AIX platform for scenarios, where 4 CPUs are used.
- The maximum number of threads available for activation specifications is increased to 30 threads. The maximum concurrency is set to 50 threads.
- Database connection pool size for the BPC DB is increased to 60, and the statement cache size for the BPC DB is increased to 300.
- The maximum number of threads that are available for JMS connection pools is set to 40.
- Connectivity to the local database uses the DB2 JDBC Universal Driver Type 2 driver.
- WebSphere Process Server JVM heap size is set to a fixed size of 1024 MB. In addition, the gencon garbage collection policy is used.

### 4.1.2 Three-tier configuration with Web services and remote DB2 server

This configuration was used in our internal performance work to evaluate the performance of a business process that models insurance claims processing. The Web services binding is used for communications. The business process has two modes of operation:

- A microflow that processes claims where no human intervention is required
- A macroflow that processes claims when a human task is required (for example, if the claim amount is above a certain limit)
The following conditions are used for the WebSphere Process Server environment:

- Use the production server template
- Tracing is disabled
- PMI is disabled
- Security disabled
- Remote (type 4) JDBC driver utilized
- Business Process support is established with bpeconfig.jacl (note that this sets the Data sources → BPEDataSourceDb2 → WebSphere Application Server data source properties statement cache to 128)

Common WebSphere Process Server server tuning includes the following conditions:

- PMI disabled
- HTTP maxPersistentRequests to -1
- Java Heap minimum and maximum set to 1024 MB
- GC policy was set to -XgcPolicy:gencon
- Remote DB2 databases (JDBC connection type 4) used for BPE, SIB System, and SIB BPC databases
- Connection pool maximum connections for SIB System database set to 30
- Connection pool maximum connections for SIB BPC database set to 50

<table>
<thead>
<tr>
<th>Value</th>
<th>Windows CPUs</th>
<th>AIX CPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Java nursery size</strong></td>
<td>256 MB</td>
<td>256 MB</td>
</tr>
<tr>
<td><strong>Default</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Web Container Thread Pool Max</strong></td>
<td>75</td>
<td>75</td>
</tr>
<tr>
<td><strong>BPE database connection pool max</strong></td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td><strong>Common database connection pool max</strong></td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td><strong>J2C connection factories</strong></td>
<td>30</td>
<td>40</td>
</tr>
<tr>
<td><strong>J2C connection factories</strong></td>
<td>30</td>
<td>40</td>
</tr>
<tr>
<td><strong>J2C connection factories</strong></td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td><strong>J2C activation specifications</strong></td>
<td>35</td>
<td>70</td>
</tr>
</tbody>
</table>
Chapter 4. Initial configuration settings

4.2 WebSphere Enterprise Service Bus settings

This section discusses settings that are used for select internal performance evaluations of WebSphere ESB and were derived using the tuning methodology and guidelines described in Chapter 3, “Performance tuning and configuration” on page 23. Consider these settings useful starting points for your use of this product. For settings that we do not list, you can use the default settings that are supplied by the product installer as a starting point. See 3.1, “Performance tuning methodology” on page 24 for a discussion on tuning methodology.

4.2.1 WebSphere ESB common settings

The following settings are good starting points, regardless of binding choices:

- Tracing is disabled
- Security is disabled
- Java heap size is fixed at 1280 MB for Windows and 1280 MB for AIX
- The gencon garbage collection policy is enabled, setting the nursery heap size to 1024 MB

For details of enableInProcessConnections, go to:

The DB2 database server has three databases that are defined for use by the WebSphere Process Server server. The database logs and table spaces are spread across a RAID array to distribute disk utilization. The database that is used for the BPC.cellname.Bus data store is not tuned. The SCA.SYSTEM.cellname.BUS database and the BPE database are tuned as follows:

- The SCA.SYSTEM.cellname.BUS database is tuned with the following commands:
  
  ```
  db2 update db cfg for sysdb using locklist 400 logbufsz 512 logfilsiz 8000
  logprimary 10 logsecond 10 maxappls 250 avg_appls 50 softmax 300 maxlocks 30
  dbheap 4800
  
  db2 alter bufferpool ibmdefaultbp size 30000
  ```

- The BPE database is tuned with the following commands:
  
  ```
  db2 update db cfg for bpedb using locklist 400 logbufsz 512 logfilsiz 8000
  logprimary 10 logsecond 10 maxappls 250 avg_appls 50 softmax 300 maxlocks 30
  dbheap 4800
  
  db2 alter bufferpool ibmdefaultbp size -1
  
  db2 update db cfg for bpedb using buffpage 64000
  ```

<table>
<thead>
<tr>
<th>Tuning</th>
<th>Windows CPUs</th>
<th>AIX CPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variations</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>J2C activation specifications</td>
<td>25</td>
<td>20</td>
</tr>
<tr>
<td>SOABenchBPELMod1_AS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Custom properties</td>
<td></td>
<td></td>
</tr>
<tr>
<td>maxConcurrency</td>
<td></td>
<td></td>
</tr>
<tr>
<td>enableInProcessConnections</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
PMI monitoring is disabled

4.2.2 WebSphere ESB settings for Web services

The WebSphere ESB settings for Web services are:

- WebContainer thread pool sizes set to a maximum 50 and a minimum 10
- WebContainer thread pool inactivity timeouts for thread pools set to 3500
- HTTP settings: maxPersistentRequests to -1

4.2.3 WebSphere ESB settings for MQ and JMS

The WebSphere ESB settings for MQ and JMS are:

- Activation specification set maximum concurrent endpoints to 30
- Queue Connection factory set the maximum connection pool size to 30
- DiscardableDataBufferSize set to 10 MB, and CachedDataBufferSize set to 40 MB
- MQ JMS Non Persistent configuration
  - Set Listener Port Max Sessions to 20
  - Set Listener Port Max Messages to 10
  - Queue Connection factory set the maximum connection pool size to 30
  - Queue Connection factory set the max session pool size to 30
- MQ JMS Persistent configuration
  - Set Listener Port Max Sessions to 30
  - Set Listener Port Max Messages to 1
  - Queue Connection factory set the maximum connection pool size to 30
  - Queue Connection factory set the max session pool size to 30
- MQ configuration
  - Set Listener Port Max Sessions to 10
  - Set Listener Port Max Messages to 5
  - Queue Connection factory set the maximum connection pool size to 30
  - Queue Connection factory set the max session pool size to 30
- Generic JMS Non Persistent configuration
  - Set Listener Port Max Sessions to 10
  - Queue Connection factory set the maximum connection pool size to 10
  - Use MQConnectionFactory class
  - Set Batch size to 10 on the QCF
  - Use Bindings mode for QM connection
- Generic JMS Persistent configuration
  - Set Listener Port Max Sessions to 30
  - Queue Connection factory set the maximum connection pool size to 30
  - Use MQXAConnectionFactory class
  - Set Batch size to 10 on the QCF
  - Use Bindings mode for QM connection
4.2.4 DB2 settings for WebSphere ESB JMS persistent scenarios

The following settings are relevant only for JMS persistent scenarios because they make use of a 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 buffer pool size correctly
- Set the connection minimum and maximum to 30
- Set the statement cache size to 40
- Use a raw partition for DB2 logs

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

4.3 WebSphere Business Monitor settings

The settings in Example 4-1 are DB2 9.1 fp4 (AIX 64 bit) WebSphere Business Monitor database settings that used for internal performance evaluations of WebSphere Business Monitor. Consider these settings as useful starting points for your use of this product. Otherwise, you can use the default settings that are supplied by the product installer as a starting point.

Example 4-1   WebSphere Business 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
db2set DB2SKIPINSERTED=ON
db2set DB2SKIPDELETED=ON
db2 update db cfg for Monitor using sortheap 4096
db2 update db cfg for Monitor using logprimary 16 logfilsiz 4096 logbufsz 128
db2 update db cfg for Monitor using auto_maint off
db2 update db cfg for Monitor using auto_tbl_maint off
db2 update db cfg for Monitor using auto_runstats off
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 BP32K size 10000
db2 connect reset
Related publications

We consider the publications that we list in this section particularly suitable for a more detailed discussion of the topics that we covered in this paper.

IBM Redbooks publications

For information about ordering these publications, see “How to get IBM Redbooks publications” on page 65. Note that some of the documents referenced here might be available in softcopy only.

- Production Topologies for WebSphere Process Server and WebSphere ESB V6, SG24-7413
- WebSphere Business Process Management V6.1.2 Production Topologies, SG24-7665

Other publications

These publications are also relevant as further information sources:

Online resources

These Web sites are also relevant as further information sources:

- WebSphere Application Server Performance
  http://www.ibm.com/software/webservers/appserv/was/performance.html
- WebSphere Application Server 6.1 information center (including Tuning Guide)
  http://www.ibm.com/software/webservers/appserv/was/library/?S_CMP=rnav
- WebSphere Process Server Version 6.1.0 information center
- WebSphere Enterprise Service Bus 6.1.0 information center
- WebSphere Business Monitor 6.1.0 information center
  http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r1mx/index.jsp
- WebSphere Adapters 6.1.0 information center
- Setting up a Data Store in the Messaging Engine
- **Best practices for tuning DB2 UDB v8.1 and its databases: A handbook for high performance**
- **DB2 Tuning Tips for OLTP Applications**
- **DB2 Information Center**
  http://publib.boulder.ibm.com/infocenter/db2i/v8/index.jsp
- **DB2 9 for Linux UNIX® and Windows Support**
  http://www.ibm.com/software/data/db2/support/db2_9
- **WebSphere Process Server V6 - Business Process Choreographer: Performance Tuning of Human Workflows Using Materialized Views**
  http://www.ibm.com/support/docview.wss?rs=2307&uid=swg27009623
- **Extending a J2CA adapter for use with WebSphere Process Server and WebSphere Enterprise Service Bus**
- **WebSphere Process Server Support**
  http://www.ibm.com/software/integration/wps/support/
- **WebSphere Enterprise Service Bus Support**
  http://www.ibm.com/software/integration/wsesb/support/
- **“The Dynamic Caching Services,” WebSphere Journal**
  http://websphere.sys-con.com/read/43309.htm
- **IBM Java Diagnostic Guide 5.0**
  http://publib.boulder.ibm.com/infocenter/javasdk/v5r0/index.jsp
- **System i Performance Capabilities Reference i5/iOS Version 5, Release 4**
- **“Diagnosing Performance Problems,” Oracle Magazine; January/February 2004**
  http://www.oracle.com/technology/oramag/oracle/04-jan/o14tech_perf.html
- **Oracle Database 10g Release 2 (10.2) Documentation Library**
  http://www.oracle.com/pls/db102/homepage
- **IBM DB2 for i home page**
  http://www.ibm.com/servers/eserver/iseries/db2
- **White papers and technical documentation for WebSphere BPM on i5/iOS**
- **Performance Management on IBM i operating systems**
- **WebSphere Process Server V6.1 – Business Process Choreographer: Performance Tuning Automatic Business Processes for Production Scenarios with DB2**
How to get IBM Redbooks publications

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications, and Additional materials, as well as order hardcopy Redbooks, 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
64-bit JVM  10

A
activation specification  25, 27, 31–33, 38
Adapter Event Delivery Table  6
ALE module  30
asynchronous invocation  2, 18
audit Logging  15
audit logging  29
automatic state transition  5

B
batch  9
bottleneck  7, 25
BPC bus  34
BPC container  28, 33
BPC database  15
BPE database  6, 15
BPEDataSource  28
BPEDB  7
BPEDB data source  33
buffer cache size  25
bufferpool size  10, 25, 44
bus  7
business activity monitoring  3
business object  4–5, 8–9, 15–16, 20–21, 31–32, 39, 49
Business Object Map primitive  19
business process  5, 10, 15
Business Process Execution Language (BPEL)  2
Business Rules Manager  7
business state machine  5
ByteArrayOutputStream  21
bytecode verification  6

C
cache  5, 34
CachedDataBufferSize  28, 34
CBE  7
CBE header  19
CEI  7, 25–26, 28–29, 36–37, 41–42
CEI data store  42
CEI Emitter  25
CEI Event Group  42
claim check pattern  9
Classic JVM  52–53
cluster  5, 7, 25
commit  16
Commit after  17
Commit before  17
Common Base Event.  See CBE
Common Event Interface (CEI)  29
CommonEventInfrastructure_Bus  34
concurrency  2, 15, 19, 31–33
concurrent objects  9
connection factory  28, 33
connection pool  2, 24–25, 28, 33
connection pool size  25
control business object  9
CPU  24–25, 31, 43
CPU capacity  10
CPU utilization  24
Custom Mediation primitive  19–20

D
dashboard  10
dashboard table  41
data buffer size  25, 34
data mining  10
Data Movement Service (DMS)  41
data source  24–25, 28, 33, 36
data store  9–10, 34–37, 42
database  2–3, 7, 10–11, 25
Database Lookup primitive  19
database management system (DBMS)  2
database partitioning  7
data-driven scaling methodology  7
DataFactory.create(Type type)  5
Datamart  10
DB2  2, 6, 23, 25, 33–37, 40, 42, 44–47, 52–53, 61
default thread pool  32
deferred-parsing  21
Derby  2, 6, 25, 34, 36, 42
development mode  6, 25
development profile  25
DiscardableDataBufferSize  28, 34
disk  25
disk subsystem  2
disk utilization  24

E
Edge component  31
E-mail adapter  40
Enable persistence and queries of business-relevant data  15
enableInProcessConnections  14
enablejvm  52
Endpoint Lookup primitive  19
event  3, 11, 21, 42–43
Event Delivery  21
event delivery  30, 40
event delivery directory  40
event emitter  41
Event Emitter primitive  19–20
event order  7
event processing  10

© Copyright IBM Corp. 2008. All rights reserved.
event processing table  41
event table  40
event-driven state transition  5

F
Fan In primitive  19
Fan Out primitive  19
file store  10, 34
Flat Files adapter  40
FTP adapter  40
full support topology  7

G
garbage collection  26, 40, 48–51
GC. See garbage collection
gencon garbage collection policy  59
Generational Concurrent Garbage Collector  27
Generational Concurrent GC  49
granularity  3

H
heap  30
heap analysis  49
heap memory  25
heap size  2, 8, 10, 25, 27, 30, 39, 48–50, 52
heap usage  51
hosts file  15
human task  2–3, 15
Human Task activity  17

I
IBM Pattern Modeling and Analysis Tool (PMAT)  49
IBM Pattern Modeling and Analysis Tool for Java Garbage Collector  49
IBM Redbooks Web site  65
Contact us  xi
IBM Technology for Java  52
IBM WebSphere Adapter for Flat Files Version 6.1  9
import component  33
initial heap setting  52
initial heap size  26, 49–50
inline human task  17
instance validation  15
Interaction Style  21
interruptible process  2–3
Invoke activity  17
iostat  24
IT monitoring  3

J
Java component  14
Java heap  8, 20
Java heap size  2, 8, 26, 39, 41, 48–49, 59
Java object  8
Java snippet  3, 14
Java Snippet activity  17
java.lang.OutOfMemory  50
JIT compilation  6
JMS binding  33
JVM  6–8, 10, 14–15, 24, 26, 30, 48–53
JVM setting  6

L
large business object  20
large object  8–10, 30
large object design pattern  9
lazy cleanup  20
listener port  25, 27, 31
localhost  14
lock escalation  46
locklist  46
log file  45–46
long running process  3
long-running process  15–18

M
macroflow  2–3
maxBatchSize  27
maxConcurrency  27, 31–33
Maximum Connections  28, 33
maximum heap setting  52
maximum heap size  10, 26, 49–50
maximum number of sessions  19
maxlock  46
maxlocks  46
MaxSession  19
MDB  27, 31–33
MDBThreadPool  32
MEDB  7
mediation  19, 21
memory  5, 8, 24–25
Message Driven Beans (MDB)  32
Message Element Setter primitive  19
Message Filter primitive  19
Message Logger primitive  19–20
message store  10
message-driven bean (MDB)  25
messaging engine  10, 24, 28, 33–37, 42
messaging engine (ME)  7
messaging engine data store  6
metadata  4–5
microflow  2–3
Monitor model  10
monitor model  10
MQ binding  19, 33
MQ JMS binding  19, 33
MQ link  19
multi-threaded SCA client  15
mySAP.com adapter  30

N
Native JDBC  53
netstat  24
calendar  6, 24
Network Deployment  7
<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>non-interruptible process</td>
<td>2–3, 15</td>
</tr>
<tr>
<td>nursery heap size</td>
<td>59</td>
</tr>
<tr>
<td>nursery size</td>
<td>25</td>
</tr>
<tr>
<td>One of N policy</td>
<td>7</td>
</tr>
<tr>
<td>one-way invocation</td>
<td>15</td>
</tr>
<tr>
<td>one-way synchronous call</td>
<td>15</td>
</tr>
<tr>
<td>Oracle</td>
<td>25, 47, 51</td>
</tr>
<tr>
<td>OutOfMemory</td>
<td>38–40, 50–51</td>
</tr>
<tr>
<td>Participates</td>
<td>17</td>
</tr>
<tr>
<td>payload</td>
<td>9, 20</td>
</tr>
<tr>
<td>Performance Monitoring Infrastructure (PMI)</td>
<td>3</td>
</tr>
<tr>
<td>Plain Old Java Objects (POJOs)</td>
<td>3</td>
</tr>
<tr>
<td>pollPeriod</td>
<td>26, 30, 38–40</td>
</tr>
<tr>
<td>pollQuantity</td>
<td>26, 30, 38–40</td>
</tr>
<tr>
<td>Preferred Interaction Style</td>
<td>18</td>
</tr>
<tr>
<td>prepared statement cache size</td>
<td>25, 28</td>
</tr>
<tr>
<td>ps</td>
<td>24</td>
</tr>
<tr>
<td>QoS qualifier</td>
<td>15</td>
</tr>
<tr>
<td>QSYSWRK</td>
<td>53</td>
</tr>
<tr>
<td>Quality of Service (QoS)</td>
<td>15, 34</td>
</tr>
<tr>
<td>queue connection factory</td>
<td>25, 33, 42</td>
</tr>
<tr>
<td>RAID</td>
<td>10</td>
</tr>
<tr>
<td>RAID adapter</td>
<td>2</td>
</tr>
<tr>
<td>RAID adapters</td>
<td>6</td>
</tr>
<tr>
<td>RAM</td>
<td>10</td>
</tr>
<tr>
<td>relationship database</td>
<td>6</td>
</tr>
<tr>
<td>request-response</td>
<td>15</td>
</tr>
<tr>
<td>Requires own</td>
<td>17</td>
</tr>
<tr>
<td>SAP adapter</td>
<td>40</td>
</tr>
<tr>
<td>SCA application bus</td>
<td>28</td>
</tr>
<tr>
<td>SCA binding</td>
<td>14</td>
</tr>
<tr>
<td>SCA import</td>
<td>18</td>
</tr>
<tr>
<td>SCA module</td>
<td>14</td>
</tr>
<tr>
<td>SCA system bus</td>
<td>28</td>
</tr>
<tr>
<td>SCA.APPLICATION bus</td>
<td>34</td>
</tr>
<tr>
<td>SCA.SYSTEM bus</td>
<td>34</td>
</tr>
<tr>
<td>schema</td>
<td>4, 35–37, 47</td>
</tr>
<tr>
<td>security</td>
<td>59</td>
</tr>
<tr>
<td>Service Component Architecture (SCA)</td>
<td>2, 14</td>
</tr>
<tr>
<td>service invocation</td>
<td>3</td>
</tr>
<tr>
<td>Set Message Type primitive</td>
<td>19</td>
</tr>
<tr>
<td>short running process</td>
<td>3</td>
</tr>
<tr>
<td>sib.msgstore.cachedDataBufferSize</td>
<td>42</td>
</tr>
<tr>
<td>sib.msgstore discardableDataBufferSize</td>
<td>42</td>
</tr>
<tr>
<td>SimpleType</td>
<td>4</td>
</tr>
<tr>
<td>SplitBySize mode</td>
<td>9</td>
</tr>
<tr>
<td>SplitCriteria</td>
<td>9</td>
</tr>
<tr>
<td>stale event</td>
<td>40</td>
</tr>
<tr>
<td>startup time</td>
<td>6</td>
</tr>
<tr>
<td>state</td>
<td>3</td>
</tr>
<tr>
<td>state machine</td>
<td>5</td>
</tr>
<tr>
<td>state transition</td>
<td>5</td>
</tr>
<tr>
<td>stopwatch-style measures</td>
<td>10</td>
</tr>
<tr>
<td>StringBuffer</td>
<td>21</td>
</tr>
<tr>
<td>Sun HotSpot JVM</td>
<td>48–49</td>
</tr>
<tr>
<td>synchronous</td>
<td>2</td>
</tr>
<tr>
<td>synchronous interaction</td>
<td>3</td>
</tr>
<tr>
<td>synchronous SCA</td>
<td>29</td>
</tr>
<tr>
<td>synchronous SCA binding</td>
<td>14</td>
</tr>
<tr>
<td>table space</td>
<td>25, 35–36, 43, 45, 47</td>
</tr>
<tr>
<td>template</td>
<td>6</td>
</tr>
<tr>
<td>thread</td>
<td>15, 17, 32–33</td>
</tr>
<tr>
<td>thread pool</td>
<td>2, 24–27, 31–33, 40, 43</td>
</tr>
<tr>
<td>Tivoli Performance Viewer</td>
<td>24</td>
</tr>
<tr>
<td>Toolbox</td>
<td>53</td>
</tr>
<tr>
<td>tracing</td>
<td>2, 20, 25–26, 29, 59</td>
</tr>
<tr>
<td>transaction</td>
<td>2–3, 15–17</td>
</tr>
<tr>
<td>transformation</td>
<td>16</td>
</tr>
<tr>
<td>two-way</td>
<td>15</td>
</tr>
<tr>
<td>variable</td>
<td>16</td>
</tr>
<tr>
<td>verbosegc</td>
<td>2, 24, 48–51</td>
</tr>
<tr>
<td>virtual resources</td>
<td>25</td>
</tr>
<tr>
<td>vmstat</td>
<td>24</td>
</tr>
<tr>
<td>Web container</td>
<td>14, 32</td>
</tr>
<tr>
<td>Web container port</td>
<td>14</td>
</tr>
<tr>
<td>Web service binding</td>
<td>14</td>
</tr>
<tr>
<td>Web service client</td>
<td>14</td>
</tr>
<tr>
<td>webcontainer</td>
<td>24</td>
</tr>
<tr>
<td>WebSphere Business Monitor dashboards</td>
<td>41</td>
</tr>
<tr>
<td>WebSphere Business Monitor database</td>
<td>10, 41, 61</td>
</tr>
<tr>
<td>WebSphere Business Monitor installation</td>
<td>41</td>
</tr>
<tr>
<td>WebSphere Business Monitor server</td>
<td>10</td>
</tr>
<tr>
<td>WebSphere MQ queue</td>
<td>19</td>
</tr>
<tr>
<td>wildcard</td>
<td>14</td>
</tr>
<tr>
<td>wire format</td>
<td>8</td>
</tr>
<tr>
<td>Work Area Service</td>
<td>29</td>
</tr>
<tr>
<td>work manager</td>
<td>24, 26, 40</td>
</tr>
<tr>
<td>XML Schema Definition (XSD)</td>
<td>4</td>
</tr>
<tr>
<td>XSLT primitive</td>
<td>19–20</td>
</tr>
</tbody>
</table>
IBM WebSphere Business Process Management V6.1 Performance Tuning

This IBM Redpaper publication provides performance tuning tips and best practices that are based on the performance team’s experience for IBM WebSphere Process Server 6.1.0, IBM WebSphere Enterprise Service Bus 6.1.0, IBM WebSphere Adapters 6.1.0, and IBM WebSphere Business Monitor 6.1.0. These products represent an integrated development and runtime environment that is based on a key set of service-oriented architecture (SOA) and business process management (BPM) technologies, including Service Component Architecture (SCA), Service Data Object (SDO), and Business Process Execution Language (BPEL) for Web Services. These technologies in turn build upon the core capabilities of the IBM WebSphere Application Server 6.1, including the Java™ Connector Architecture (JCA) V1.5 on which WebSphere Adapters V6.1.0 are based.

This paper discusses the performance implications of the supporting runtime environment, and relates a subset of best practices as well as tuning and configuration parameters for the different software technologies that are involved. The audience for this paper includes a wide variety of groups such as customers, services, technical marketing, and development. Note that this paper is not as comprehensive as a tuning, sizing, or capacity planning guide, although it serves as a useful reference for these activities.

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

For more information:
ibm.com/redbooks