

# WebSphere Business Process Management 6.2.0 Performance Tuning

Learn valuable tips for tuning

Get the latest best practices

Try the example settings



IBM Business Process Management  
Performance Teams





International Technical Support Organization

**WebSphere Business Process Management 6.2.0  
Performance Tuning**

July 2009

**Note:** Before using this information and the product it supports, read the information in “Notices” on page vii.

**First Edition (July 2009)**

This edition applies to WebSphere Process Server V6.2, WebSphere Enterprise Service Bus 6.2, WebSphere Integration Developer V6.2, WebSphere Business Monitor V6.2, and WebSphere Business Services Fabric V6.2.

© Copyright International Business Machines Corporation 2009. 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

<b>Notices</b> .....	vii
Trademarks .....	viii
<b>Preface</b> .....	ix
Products covered by this publication .....	ix
The team that wrote this paper .....	x
Become a published author .....	xi
Comments welcome .....	xi
<b>Chapter 1. Architecture best practices</b> .....	1
1.1 Top tuning and deployment guidelines .....	2
1.2 Modeling .....	3
1.2.1 Choose non-interruptible processes whenever possible .....	3
1.2.2 Choose query tables for task and process list queries .....	3
1.2.3 Choose the appropriate granularity for a process .....	4
1.2.4 Use events judiciously .....	4
1.2.5 Choose efficient meta-data management .....	4
1.2.6 Choose business processes over business state machines if possible .....	6
1.2.7 Minimize state transitions in BSM .....	6
1.3 Topology .....	6
1.3.1 Deploy appropriate hardware .....	6
1.3.2 Use a high-performing database (such as DB2) .....	7
1.3.3 Deploy local modules in the same server .....	7
1.3.4 Best practices for clustering .....	7
1.3.5 Evaluate service providers and external interfaces .....	9
1.4 Large objects .....	10
1.4.1 Factors affecting large object size processing .....	10
1.4.2 Large object design patterns .....	11
1.5 64-bit considerations .....	12
1.6 WebSphere Business Monitor .....	12
1.6.1 Event processing .....	13
1.6.2 Dashboard .....	13
1.6.3 Database server .....	13
<b>Chapter 2. Development best practices</b> .....	15
2.1 Service Component Architecture (SCA) considerations .....	16
2.1.1 Cache results of ServiceManager.locateService() .....	16
2.1.2 Reduce the number of SCA modules, when appropriate .....	16
2.1.3 Use synchronous SCA bindings across local modules .....	16
2.1.4 Utilize multi-threaded SCA clients to achieve concurrency .....	16
2.1.5 Add quality of service qualifiers at appropriate level .....	16
2.2 Business process considerations .....	16
2.2.1 Modeling best practices for activities in a business process .....	17
2.2.2 Avoid 2-way synchronous invocation of long-running processes .....	17
2.2.3 Minimize number and size of BPEL variables and BOs .....	18
2.3 Human task considerations .....	18
2.4 Business process and human tasks client considerations .....	18
2.5 Transactionality considerations .....	19
2.5.1 Exploit SCA transaction qualifiers .....	19

2.5.2	Use transactional attributes for activities in long-running processes . . . . .	20
2.6	Invocation style considerations . . . . .	20
2.6.1	Use asynchrony judiciously. . . . .	21
2.6.2	Set the preferred interaction style to sync whenever possible . . . . .	21
2.6.3	Minimize cross-component asynchronous invocations within a module. . . . .	21
2.7	Messaging considerations. . . . .	22
2.7.1	Choose MQ or MQ/JMS binding rather than MQ Link . . . . .	22
2.7.2	Set MaxSession for the MQ/JMS export binding . . . . .	22
2.7.3	Use mediations that benefit from WebSphere ESB optimizations . . . . .	22
2.7.4	Usage of XSLTs versus BO maps . . . . .	23
2.7.5	Configure WebSphere ESB resources . . . . .	23
2.8	Large object best practices . . . . .	24
2.8.1	Avoid lazy cleanup of resources . . . . .	24
2.8.2	Avoid tracing when processing large BOs . . . . .	24
2.8.3	Avoid buffer-doubling code . . . . .	24
2.8.4	Make use of deferred-parsing friendly mediations for XML docs . . . . .	24
2.9	WebSphere InterChange Server migration considerations. . . . .	25
2.10	Adapters: Configure synchronous event delivery . . . . .	25
2.11	WebSphere Integration Developer considerations . . . . .	26
2.11.1	Leverage hardware advantages . . . . .	26
2.11.2	When appropriate, tune the server for fast publish. . . . .	26
2.11.3	Make use of shared libraries in order to reduce memory consumption . . . . .	26
2.12	Fabric considerations . . . . .	26
2.12.1	Only specify pertinent context properties in context specifications. . . . .	27
2.12.2	Bound the range of values for context keys . . . . .	27
<b>Chapter 3.</b>	<b>Performance tuning and configuration . . . . .</b>	<b>29</b>
3.1	Performance tuning methodology . . . . .	30
3.2	Tuning checklist . . . . .	31
3.3	Tuning parameters . . . . .	32
3.3.1	Tracing and logging flags . . . . .	32
3.3.2	Java tuning parameters . . . . .	33
3.3.3	MDB ActivationSpec . . . . .	34
3.3.4	MQ listener port . . . . .	34
3.3.5	Thread pool sizes . . . . .	34
3.3.6	JMS connection pool sizes . . . . .	35
3.3.7	Data source connection pool size. . . . .	35
3.3.8	Data source prepared statement cache size. . . . .	35
3.3.9	Utilize non-XA data sources for CEI data, if possible . . . . .	35
3.3.10	Messaging engine properties . . . . .	35
3.3.11	Run production servers in production . . . . .	36
3.4	Advanced tuning . . . . .	36
3.4.1	Tracing and monitoring considerations . . . . .	36
3.4.2	Tuning for large objects . . . . .	36
3.4.3	Tuning for maximum concurrency. . . . .	37
3.4.4	Messaging tuning . . . . .	41
3.4.5	Web services tuning . . . . .	44
3.4.6	Business Process Choreographer tuning . . . . .	45
3.4.7	WebSphere ESB tuning . . . . .	46
3.4.8	WebSphere adapters tuning . . . . .	47
3.4.9	WebSphere Business Monitor tuning . . . . .	49
3.4.10	Database: general tuning . . . . .	51
3.4.11	Database: DB2-specific tuning . . . . .	51

3.4.12 Database: Oracle-specific tuning . . . . .	55
3.4.13 Advanced Java heap tuning . . . . .	56
3.4.14 Power management tuning . . . . .	60
3.4.15 Tuning for WebSphere InterChange Server migrated applications. . . . .	60
3.4.16 WebSphere Business Services Fabric tuning . . . . .	60
3.4.17 IBM i tuning . . . . .	60
<b>Chapter 4. Initial configuration settings . . . . .</b>	<b>63</b>
4.1 WebSphere Process Server settings . . . . .	64
4.1.1 Two-tier configuration using JMS file store . . . . .	64
4.1.2 Three-tier configuration with Web service and remote DB2 server. . . . .	65
4.2 WebSphere ESB settings . . . . .	68
4.2.1 WebSphere ESB common settings. . . . .	68
4.2.2 WebSphere ESB settings for Web services . . . . .	68
4.2.3 WebSphere ESB settings for MQ and JMS . . . . .	68
4.2.4 DB2 settings for WebSphere ESB JMS persistent scenarios . . . . .	69
4.3 WebSphere Business Monitor database settings . . . . .	69
<b>Related publications . . . . .</b>	<b>71</b>
IBM Redbooks publications . . . . .	71
Online resources . . . . .	71
How to get Redbooks. . . . .	72
Help from IBM . . . . .	73





# 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 about 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®  
alphaWorks®  
DB2®  
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.

Interchange, and the Shadowman logo are trademarks or registered trademarks of Red Hat, Inc. in the U.S. and other countries.

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, JSP, JVM, Power Management, 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.

Intel Core2 Duo, Intel, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries 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® BPM performance teams. It provides performance tuning tips and best practices for the following products:

- ▶ WebSphere Process Server 6.2.0
- ▶ WebSphere Enterprise Service Bus 6.2.0
- ▶ WebSphere Integration Developer 6.2.0
- ▶ WebSphere Business Monitor 6.2.0
- ▶ WebSphere Business Services Fabric 6.2.0

These products represent an integrated development and runtime environment based on a key set of service-oriented architecture (SOA) and Business Process Management (BPM) technologies:

- ▶ Service Component Architecture (SCA)
- ▶ Service Data Object (SDO)
- ▶ Business Process Execution Language for Web Services (BPEL)

These technologies in turn build on the core capabilities of the WebSphere Application Server 6.1.

For those who either are considering or are in the very early stages of implementing a solution incorporating these products, this publication provides best practices for application development and deployment, and setup, tuning, and configuration information. It provides a useful introduction to many of the issues influencing each product's performance, and could act as a guide for making rational first choices in terms of configuration and performance settings.

Finally, these products build on the capabilities of WebSphere Application Server, so consult tuning, configuration, and best practices information for WebSphere Application Server and corresponding platform Java™ Virtual Machines (JVMs) (documented in “Related publications” on page 71).

## Products covered by this publication

Below is a short description of each product covered in this publication:

- ▶ WebSphere Process Server allows the deployment of standards-based business integration applications in a service-oriented architecture, which takes everyday business applications and breaks them down into individual business functions and processes, rendering them as services. Based on the robust J2EE™ 1.4 infrastructure and associated platform services provided by WebSphere Application Server, WebSphere Process Server can help you meet current business integration challenges. This includes, but is not limited to, business process automation.
- ▶ WebSphere Enterprise Service Bus provides the capabilities of a standards-based enterprise service bus. WebSphere ESB manages the flow of messages between service requesters and service providers. Mediation modules within WebSphere ESB handle mismatches between requesters and providers, including protocol or interaction-style, interface, and quality of service mismatches.
- ▶ WebSphere Integration Developer is the development environment for building WebSphere BPM solutions. It is a common tool for building service-oriented

architecture-based integration solutions across WebSphere Process Server, WebSphere Enterprise Service Bus, and other WebSphere BPM products.

- ▶ WebSphere Business Monitor provides the ability to monitor business processes in real-time, providing a visual display of business process status, business performance metrics, and key business performance indicators, together with alerts and notifications to key users that enables continuous improvement of business processes.
- ▶ WebSphere Business Services Fabric provides an end-to-end platform for the rapid assembly, delivery, and governance of industry-focused composite business services in an SOA solution. It adds an industry-specific layer to the IBM SOA Foundation by enabling dynamic business service personalization and delivery based on business context.

## Publication structure

The first three chapters of this publication are about *best practices and tuning considerations* for three different phases of WebSphere BPM projects:

- ▶ Architecture
- ▶ Development
- ▶ Deployment

At least one of these chapters will be of interest to any reader of this publication, and many will find value in all three chapters. There is a list of key tuning and deployment guidelines in 1.1, “Top tuning and deployment guidelines” on page 2. We strongly urge all readers to take note of this list since the authors have seen numerous instances where this information is very useful. Chapter 4, “Initial configuration settings” on page 63, describes configuration options for representative performance workloads, and the publication concludes with a list of useful references in “Related publications” on page 71. Below is a summary of each chapter:

- ▶ Chapter 1, “Architecture best practices” on page 1: recommendations for architecture and topology decisions that will produce well-performing and scalable solutions
- ▶ Chapter 2, “Development best practices” on page 15: guidelines for solution developers that will lead to high-performing systems
- ▶ Chapter 3, “Performance tuning and configuration” on page 29: a discussion of the configuration parameters and settings for the major software components that comprise a business process management solution
- ▶ Chapter 4, “Initial configuration settings” on page 63: details of the software configurations used for representative workloads used by the IBM performance team working on these products
- ▶ “Related publications” on page 71: links to best practices, performance information, and product information for both the products discussed in this publication, and related products such as WebSphere Application Server, DB2®, and so on

## The team that wrote this paper

This publication was produced by the following members of the IBM WebSphere Business Process Management Performance Team, located in Austin, Texas; Böblingen, Germany; Hursley, England; and Rochester, Minnesota:

- ▶ Phani Achanta
- ▶ Rajiv Arora
- ▶ Mike Collins
- ▶ Gudrun Eckl-Regenhardt
- ▶ Steve Garde

- ▶ Tabitha Gichora
- ▶ Jonas Grundler
- ▶ Weiming Gu
- ▶ Paul Harris
- ▶ Steve Heuer
- ▶ Ben Hoflich
- ▶ Telford Knox
- ▶ Kean Kuiper
- ▶ Sam Massey
- ▶ Thomas Muehlfriedel
- ▶ Rachel Norris
- ▶ Sefika Prcic
- ▶ Chris Richardson
- ▶ Randall Theobald

## 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](http://ibm.com/redbooks/residencies.html)

## Comments welcome

Your comments are important to us!

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

- ▶ Use the online **Contact us** review Redbooks form found at:

[ibm.com/redbooks](http://ibm.com/redbooks)

- ▶ Send your comments in an e-mail to:

[redbooks@us.ibm.com](mailto: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 on how to architect a high-performing and scalable WebSphere Business Process Management (BPM) solution. The purpose of this chapter is to highlight the best practices specifically associated with the technologies and features delivered in the WebSphere BPM products covered in this paper. These products are built on top of existing technologies like WebSphere Application Server, Platform Messaging, and DB2. Each of these technologies has associated best practices that apply. It is not our intent to enumerate in this paper. Instead, the reader is referred to “Related publications” on page 71 for a set of references and pointers to this information.

## 1.1 Top tuning and deployment guidelines

The remainder of this chapter details architectural best practices for all key components. Development best practices and performance tuning and configuration are covered in subsequent chapters. We strongly encourage you to read these chapters, since we have found this information to be very beneficial for numerous customers over the years. However, if you read nothing else in this publication, read and adhere to the following key tuning and deployment guidelines, since 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) will be required on the tiers that host the message and data stores to achieve acceptable performance. This point cannot be overstated. 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 will greatly help in determining the optimal settings. Further information is available in “Java tuning parameters” on page 33.
- ▶ 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 must be persisted). However, there is significant performance overhead associated with macroflows. For details see “Choose non-interruptible processes whenever possible” on page 3.
- ▶ Use DB2 instead of the default Derby database management system (DBMS). DB2 is a high-performing, industrial strength database designed to handle high levels of throughput and concurrency. It scales well and 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 “Database: general tuning” on page 51.
- ▶ Disable tracing. Tracing is clearly important when debugging, but the overhead of tracing severely impacts performance. More information is available in “Tracing and monitoring considerations” on page 36.
- ▶ Configure thread and connection pools for sufficient concurrency. This is especially important for high-volume, highly concurrent workloads, since the thread pool settings directly influence how much work can be concurrently processed by the server. For more information see “Configure thread pool sizes” on page 39.
- ▶ For task and process list queries, use composite query tables. Query tables are designed to produce excellent response times for high-volume task and process list queries. For details see “Choose query tables for task and process list queries” on page 3.
- ▶ Use work-manager-based navigation to improve throughput for long-running processes. This optimization reduces the number of objects allocated, the number of objects retrieved from the database, and the number of messages sent for Business Process Choreographer messaging. For further information see “Tuning WorkManager-based navigation for business processes” on page 45.
- ▶ Avoid unnecessary usage of asynchronous invocations. Asynchronous invocation is often needed on the edges of modules, but not within a module. Utilize synchronous preferred interaction styles, as is described in “Set the preferred interaction style to sync whenever possible” on page 21.
- ▶ Avoid too granular of transaction boundaries in Service Component Architecture (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 “Transactionality considerations” on page 19.

## 1.2 Modeling

This section describes best practices for modeling.

### 1.2.1 Choose non-interruptible processes whenever possible

Use interruptible processes (that is, 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 typically executed in several J2EE transactions, requiring that state be persisted in a database at transaction boundaries.

Whenever possible, utilize synchronous interactions for non-interruptible processes, since this communication style is generally more efficient than asynchronous interactions.

A process is interruptible if the Process is long-running check box is set in WebSphere Integration Developer via Properties → Details 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 query tables for task and process list queries

Query tables are introduced in WebSphere Process Server 6.2.0. Query tables are designed to provide good response times for high-volume task list and process list queries. Query tables offer improved query performance:

- ▶ Improved access to work items reduces the complexity of the database query.
- ▶ Configurable high-performance filters on tasks, process instances, and work items allow for efficient filtering.
- ▶ Composite query tables can be configured to bypass authorization through work items.
- ▶ Composite query tables allow the definition of query tables that reflect the information that is displayed on task lists and process lists presented to users.

For further information see the references below:

- ▶ WebSphere Process Server - Query Table Builder  
<http://www.ibm.com/support/docview.wss?uid=swg24021440>
- ▶ Query Tables in Business Process Choreography publication in the WebSphere Process Server 6.2.0 Info Center  
[http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r2mx/topic/com.ibm.websphere.bpc.620.doc/doc/bpc/c6bpe1\\_querytables.html](http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r2mx/topic/com.ibm.websphere.bpc.620.doc/doc/bpc/c6bpe1_querytables.html)

### 1.2.3 Choose the appropriate granularity for a process

A business process and its individual steps should have *business significance* and not try to mimic programming-level granularity. Use programming techniques like Plain Old Java Objects (POJOs) or Java snippets for logic without business significance. This topic is discussed further in the *Software components: coarse-grained versus fine-grained* paper available at:

<http://www.ibm.com/developerworks/library/ws-soa-granularity/index.html>

### 1.2.4 Use events judiciously

The purpose of Common Base Event (CBE) emission in WebSphere Process Server is for business activity monitoring. Since CBE emission uses a persistent mechanism, it is inherently heavy weight. One should utilize CBE only for events that have business relevance. Furthermore, we do not recommend emitting CBEs to a database. Instead, CBE emission should be done via the messaging infrastructure. Finally, do not confuse business activity monitoring and IT monitoring. The Performance Monitoring Infrastructure (PMI) is far more appropriate for the latter.

With this in mind, the following generally is true 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, this is fairly natural. Use events to track when long-running activities complete, when human tasks change state, and so on.
- ▶ For short-running flows that complete within seconds, it is usually sufficient to know that a flow completed, 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.

### 1.2.5 Choose efficient meta-data management

This section describes best practices for designing meta-data usage.

#### **Follow Java language specification for complex DataType names**

While WebSphere Process Server allows characters in business object (BO) type names that would not be permissible in Java class names (the underscore (\_), for example), the internal data representation of complex data type names does make use of Java types. As such, performance is better if BO 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 XSDs

Some XML schema definition (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 explicitly declared, then a new sub-type (a derived type) is generated at runtime. Performance is generally better if this can be avoided, so avoid adding restrictions to elements of the primitive type where possible. If a restriction is unavoidable, consider creating a new, concrete SimpleType that extends the primitive type to include the restriction. Then XSD elements may utilize that type without degraded performance.

## Avoid referencing elements from one XSD in another XSD

If A.xsd defines an element, AElement (shown in Example 1-1), it may be referenced from another file, B.xsd (shown in Example 1-2).

### Example 1-1 AElement XSD

---

```
<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>
```

---

### Example 1-2 AElement referenced from another file

---

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

---

This has been shown to perform poorly. It is much better to define the type concretely and then make any new elements use this type. So, A.xsd becomes what Example 1-3 shows.

### Example 1-3 AElementType XSD

---

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

---

B.xsd becomes what Example 1-4 shows.

### Example 1-4 BElement XSD

---

```
<xs:element name="BElement" type="AElementType" minOccurs="0" />
```

---

## Reuse data object type metadata where possible

Within application code, it is common to refer to types, for instance, when creating a new business object. It is possible to refer to a business object type by name, for 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, via `DataObject.getType()`) and reuse that type for the second and future uses.

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

Business state machines (BSM) 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. Furthermore, WebSphere Process Server 6.1.0 (and subsequent releases) delivered significant performance improvements for BSM applications. However, BSM is implemented using the business process infrastructure, so there will always be a performance impact when choosing BSM over business processes. If an application can be modeled using either BSM 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 BSM performance.

## 1.2.7 Minimize state transitions in BSM

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 BSM 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 this into consideration when designing a BSM.

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

# 1.3 Topology

In this section we discuss choosing an appropriate topology for your solution.

## 1.3.1 Deploy appropriate hardware

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

- Processor cores

Ensure that WebSphere Process Server and WebSphere ESB are installed on a modern server system with multiple processor cores. WebSphere Process Server and WebSphere ESB scale well, both vertically in terms of symmetric multiprocessing (SMP) scaling, and horizontally in terms of clustering.

- Memory

WebSphere Process Server and WebSphere ESB benefit from both a robust memory subsystem and 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 that there is enough physical memory for all the applications (JVMs) combined that are expected to run concurrently on the system. A rough rule of thumb is 2 GB per WebSphere Process Server/WebSphere ESB JVM.

- ▶ Disk

Ensure that the systems hosting the message and data stores, typically the database tiers, have fast storage. This means utilizing 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.

- ▶ Virtualization

Take care when using virtualization such as AIX® dynamic logical partitioning or VMWare virtual machines. Ensure that sufficient processor, memory, and I/O resources are allocated to each virtual machine or LPAR. Avoid over-committing resources.

### 1.3.2 Use a high-performing database (such as DB2)

WebSphere Process Server, WebSphere ESB, Monitor, and WebSphere Adapters (WAs) are packaged with the Derby database, an open source database designed for ease-of-use and platform neutrality. If performance and reliability are important, use an industrial strength database (such as IBM DB2) for any performance measurement or production installation. Examples of databases that can be moved to DB2 include the BPE database, relationship databases, the WebSphere Platform Messaging (WPM) Messaging Engine data stores, Adapter Event Delivery Tables, and any adapter-specific databases.

The conversion requires that the administrator create new JDBC™ providers in the administrative console under Resources → JDBC Providers. Once created, a data source can be added to connect to a database using the new provider.

### 1.3.3 Deploy local modules in the same server

If planning to deploy modules on the same physical server, better performance will be achieved by deploying the modules to the same application server JVM, as this allows the server to exploit this locality.

### 1.3.4 Best practices for clustering

We highly recommend reading *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.

It is not the intent of this section to repeat any content from the above. Rather, we will distill 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 (also known as the Golden Topology) prescribes the use of separate clusters for applications, messaging engines, and support applications like the Common Event Infrastructure (CEI) server and the Business Rules Manager. As explained in the next section, this 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 the reader is familiar with these kinds of system trade-offs, as they occur in most server middleware.

## Single instance versus clustered topology considerations

In general, there are two primary things to consider when evaluating moving to a clustered topology from a single server configuration:

- ▶ Scalability/load balancing in order to improve overall performance and throughput
- ▶ High availability/failover to prevent loss of service due to hardware or software failures

Although not mutually exclusive, there are considerations applicable to each. Most single server workloads that are driving resources to saturation would benefit to at least to some degree by moving to a clustered configuration. Points to consider when partitioning the system include:

- ▶ Performance versus high-availability requirements. A vertical clustering topology (partitioning the target single SMP server into multiple partitions/cluster nodes) does not provide a complete failover configuration.
- ▶ If we assume a vertical clustering topology, what would be the ideal node size? For example, with eight cores available, would four (2-core) partitions or two (4-core) partitions be the best solution?
- ▶ Support (for example, CEI) and messaging clusters, as well as the required databases and database configuration, play a large part in maintaining balance and scalability for clustered configurations. This becomes particularly important when an application is characterized by a number of asynchronous invocations, or one that may be implementing a complete BPM scenario that includes WB Monitor with corresponding process event emission.

## Apply a data-driven scaling methodology

The reason for deploying a clustered topology is to be able to 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 Network Deployment infrastructure of WebSphere Process Server provides this capability. However, effective scaling still requires standard performance monitoring and bottleneck analysis techniques to be used.

Here are some considerations. In the discussion below, we assume that additional cluster members imply additional server hardware:

- ▶ If deploying more than one cluster member (JVM) on a single physical system, it is important to monitor not just the resource utilization (processor core, disk, network, and so on) of the system as a whole, but also the utilization by each cluster member. This 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, there are some additional considerations:
  - A messaging engine in a cluster with *One of N* policy (to preserve event ordering) may 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 Message 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 the cluster policy can be changed to allow workload balancing with partitioned destinations.
  - A database (DB) server may become the bottleneck. Approaches to consider are:
    - If the DB server is hosting multiple DBs that are active (for example, the BPEDB and the MEDB), consider hosting each DB on a separate server.
    - If a single DB is driving load, consider a more powerful DB server.
    - Beyond the above, database partitioning and clustering capabilities can be exploited.

### 1.3.5 Evaluate service providers and external interfaces

One of the typical usage patterns for WebSphere Process Server is as an integration layer between incoming requests and backend systems for the business (target applications or service providers). In these scenarios, the throughput will be limited by the layer with the lowest throughput capacity. Considering the simple case where there is only one target application, the WebSphere Process Server based integration solution cannot achieve throughput rates higher than the throughput capacity of the target application regardless of the efficiency of the WebSphere Process Server based implementation or the size or speed of the system hosting WebSphere Process Server. Thus, it is critical to understand the throughput capacity of all target applications and service providers, and apply this information when designing the end-to-end solution.

There are two key aspects of the throughput capacity of a target application or service provider:

- ▶ Response time, both for typical cases and exception cases
- ▶ Number of requests that the target application can process at the same time (concurrency)

If each of these performance aspects of the target applications can be established, then a rough estimate of the maximum throughput capacity can be calculated. Similarly, if average throughput is known, then either one of these two aspects can be roughly calculated as well. For example, a target application that can process 10 requests per second with an average response time of 1 second can process approximately 10 requests at the same time (throughput / response time = concurrency).

The throughput capacity of target applications is critical to projecting the end-to-end throughput of an entire application. Also, the concurrency of target applications should be considered when tuning the concurrency levels of the upstream WebSphere Process Server based components. For example, if a target application can process 10 requests at the same time, the WebSphere Process Server components that invoke this application should be tuned so that the simultaneous requests from WebSphere Process Server at least match the

concurrency capabilities of the target. Additionally, overloading target applications should be avoided since such configurations will not result in any increase in overall application throughput. For example, if 100 requests are sent to a target application that can only process 10 requests at the same time, no throughput improvement will be realized versus tuning so that the number of requests made matches the concurrency capabilities of the target.

Finally, for service providers that may take a long time to reply, either as part of main line processing or in exception cases, do not utilize synchronous invocations that require a response. This is to avoid tying up the WebSphere Process Server business process and its resources until their service provider replies.

## 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 and practical guidelines for the current releases of these products.

The single most important factor affecting large object processing is the JVM.

WebSphere BPM V6.1.0 and later use the Java5 JVM, which has very different characteristics for large objects from the 1.4.2 JVM used in previous versions of WebSphere BPM. The discussion in this section is relevant for WebSphere BPM V6.1.0 and later. If you are using WebSphere BPM V6.0.2 or earlier, consult *WebSphere Business Integration V6.0.2 Performance Tuning*, REDP-4304.

In general, objects 5 MB or larger may be considered *large* and 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 successfully processed.

In order 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. Further details of heap sizes are given later in this section, but 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 BO that consumes 10 MB on an input JMS message queue may 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 BO flows through



the adapters and WebSphere Process Server or WebSphere ESB. There are a number of factors that affect the in-memory expansion of BOs.

- The single-byte binary wire representation is generally converted to multi-byte character representations (for example, Unicode), resulting an expansion factor of 2.
  - The BO may 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 BO flows through the system, it may be modified or copied, and multiple copies may 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 BO copies in order for the transaction to complete successfully.
- Number of concurrent objects being processed

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

## 1.4.2 Large object design patterns

There are two proven design patterns for processing large objects successfully. Each is described below. In cases where neither can be applied, 64-bit mode should be considered. See the next section for details.

### **Batched inputs: Send large objects as multiple small objects**

If a large object must be processed then the solutions 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 as assumed above, the solution is to group the smaller objects into conglomerate objects less than 1 MB in size. This has been done at a variety of customer sites 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 certain adapters like the Flat Files JCA Adapter can be configured to use a SplitBySize mode with a SplitCriteria set to the size of each individual object. In this case a large object would be split in chunks of the size specified by SplitCriteria to reduce peak memory usage.

### **Claim check pattern: when a small portion of an input message is used**

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

1. Detach the data payload from the message.
2. Extract the required attributes into a smaller *control* BO.

3. Persist the larger data payload to a datastore and store the *claim check* as a reference in the control BO.
4. Process the smaller control BO, which has a smaller memory footprint.
5. At the point where the solution needs the entire large payload again, check out the large payload from the datastore using the key.
6. Delete the large payload from the datastore.
7. Merge the attributes in the control BO with the large payload, taking the changed attributes in the control BO 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 BO. This approach suffers from the disadvantage of being limited to certain export/import bindings (JMS, MQ, HTTP, and JCA adapters like flat files, e-mail, and FTP) and the full payload still must be allocated in the JVM.

## 1.5 64-bit considerations

Since WebSphere Process Server 6.1.0, full 64-bit support has been available. However, applications can continue to be run in either 32-bit or 64-bit mode. In 32-bit mode, the maximum heap size is limited by the 4 GB address space size, and in most 32-bit operating systems the practical limit varies between 1.5–2.5 GB. In contrast, while maximum heap size is essentially limitless in 64-bit mode, standard Java best practices still apply. The sum of the maximum heap sizes of all the Java processes running on a system should not exceed the physical memory available on the system.

Here are the factors to consider when determining which of these modes to run in:

- ▶ If you are running a 32-bit application today and its liveset, or bytes in use after garbage collection (GC), fits well within a 1.5–2.5 GB heap, then we recommend continuing to run in 32-bit mode. Moving to 64-bit may cause a degradation in throughput and an increase in liveset.
- ▶ An excellent choice for applications whose liveset approaches or exceeds the 32-bit limits is 64-bit mode. Such applications either experience `OutOfMemoryExceptions` or suffer excessive time in GC. We consider anything greater than 10% of time in GC as excessive. These applications will enjoy much better performance when allowed to run with the larger heaps that they need. However, it is important to understand that 64-bit mode does increase liveset due to the overall growth in storage when using 64-bit pointers. There must always be sufficient physical memory on the system to back the large heap.
- ▶ Also, 64-bit mode is a good choice for applications that, though well behaved on 32 bit, could be algorithmically modified to perform much better with larger heaps. An example would be an application that frequently persists data to a data store to avoid maintaining a very large in-memory cache, even if such a cache would greatly improve throughput. Recoding such an application to trade off the more space available in 64-bit heaps for less execution time would yield much better performance.

## 1.6 WebSphere Business Monitor

This section describes best practices for the WebSphere Business Monitor.

### **1.6.1 Event processing**

A major factor in event processing performance is the tuning of the Monitor Database. Attention should be paid especially to adequate bufferpool sizes to minimize disk reading activity and the placement of the database logs, which ideally should be on a physically separate disk subsystem from the database tablespaces.

### **1.6.2 Dashboard**

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

### **1.6.3 Database server**

Both event processing and Dashboard rely on a fast, well-tuned database server for good performance. The design of Monitor assumes that any customer using it has strong on-site DB administrator skills. We strongly recommend that the database tuning advice and recommendations beginning in 3.4.10, “Database: general tuning” on page 51 be read and followed.





## 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. WebSphere Integration Developer is used to implement the vast majority of these best practices.

## 2.1 Service Component Architecture (SCA) considerations

This section discusses the Service Component Architecture 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, as service location is a relatively expensive operation. Note that WebSphere Integration Developer-generated code does not do this, so editing is required to cache the `locateService` result.

### 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 has been optimized for module locality and will outperform 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 ESB servers.

### 2.1.4 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.

### 2.1.5 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. Since 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

This section discusses various business process considerations.

## 2.2.1 Modeling best practices for activities in a business process

Use the following guidelines when modeling activities for a business process:

- ▶ 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.
- ▶ For long-running processes, disable the “Enable persistence and queries of business-relevant data” flag under the **Properties** → **Server** tab, for both *Process* and for each individual Business Process Execution Language for Web Services (BPEL) activity. Enabling this flag causes details and history of the execution of this activity to be stored in the BPC database. This increases the load on the database and the amount of data stored for each process instance. This setting should be used only if this specific information will need to be retrieved later.
- ▶ For long-running processes, a setting of *participates* on all activities generally provides the best throughput performance. See “Use transactional attributes for activities in long-running processes” on page 20 for more details.
- ▶ Human tasks can be specified in business processes (for example, process administrators), invoke activities, and receive activities. Specify these tasks only if needed. Also, when multiple users are involved use group work items (people assignment criterion: Group) instead of individual work items for group members (people assignment criterion: Group Members).

## 2.2.2 Avoid 2-way synchronous invocation of long-running processes

When designing long-running business process components, ensure that callers of a 2-way (request/response) interface do not use synchronous semantics, as this ties up the caller's resources (thread, transaction, and so on) until the process completes. Instead, such processes should either be invoked asynchronously, or via a 1-way synchronous call, where no response is expected.

In addition, calling a 2-way interface of a long-running business process synchronously introduces difficulties when exceptions occur. Suppose that a non-interruptible process calls a long-running process using the 2-way request/response semantics and the server fails after the long-running process has completed, but before the caller's transaction is committed:

- ▶ If the caller was started by a persistent message, 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, since it was committed before the server failure. As a result, the long-running process on the server is executed twice. This duplication will cause functional problems in the application unless corrected manually.
- ▶ If the caller was not started by a persistent message, and the response of the long-running process was not submitted yet, it will end in the failed event queue.

### 2.2.3 Minimize number and size of BPEL variables and BOs

Follow these guidelines when defining BPEL variables and business objects (BOs):

- ▶ Use as few variables as possible and minimize the size and the number of BOs used. In long-running processes, each commit saves modified variables to the database (to save context), and multiple variables or large BOs make this very costly. Smaller BOs are also more efficient to process when emitting monitor events.
- ▶ Specify variables as data type variables. This improves runtime performance.
- ▶ Use transformations (maps or assigns) to produce smaller BOs by only mapping fields necessary for the business logic.

## 2.3 Human task considerations

Adhere to these guidelines when developing human tasks:

- ▶ Use group work items for large groups (people assignment criterion: Group) instead of individual work items for group members (people assignment criterion: Group Members). This results in fewer physical work items created when the application is used at runtime.
- ▶ Where possible, use native properties on the task object rather than custom properties. For example, use the priority field instead of creating a new custom property *priority*.
- ▶ Set the transactional behavior to *commit after* if the task is not part of a page-flow.

## 2.4 Business process and human tasks client considerations

The following are topics to consider when developing business process and human task clients:

- ▶ APIs that provide task details and process details, such as “`htm.getTask()`”, should not be called frequently. Use these methods only when required to display the task details of a single task, for instance.
- ▶ Do not put too much work into a single client transaction:
  - In servlet applications, a global transaction is typically not available. If the servlet calls the HTM and BFM APIs directly, transaction size is typically not a concern.
  - In EJB™ applications, make sure that transactions are not too time consuming. Long-running transactions create long-lasting locks in the database, which prevent other applications and clients from continuing processing.
- ▶ Choose the protocol that best suits your needs:
  - In a J2EE environment, use the HTM and BFM EJB APIs.
  - In a Web 2.0 application, use the REST API.
  - In an application that runs remotely from the process container, the Web services API may be an option.

Clients that follow a page-flow pattern should consider the following:

- ▶ Use the “`completeAndClaimSuccessor()`” API if possible. This provides optimal response time.
- ▶ Do not put asynchronous invocations between two steps of a page-flow, because the response time of asynchronous services increases as the load on the system increases.



- ▶ Where possible, do not use sub-process invocations between two steps of a page-flow, because sub-processes are invoked using asynchronous messaging.

Clients that present task lists and process lists to the user should consider the following:

- ▶ Use query tables for task list and process list queries. See “Choose query tables for task and process list queries” on page 3 for further information.
- ▶ Do not loop over the tasks displayed in the task or process list and execute an additional remote call for each object. This prevents the application from providing good response times and good scalability.
- ▶ Design the application such that during task list and process list retrieval, all information is retrieved from a single query table. For instance, do not make calls to retrieve the input message for task list or process list creation.

## 2.5 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 should carefully design their desired transaction boundaries as dictated by the application's needs. Transaction boundaries are expensive in system resources. Hence, the objective of this section is to guide the modeler in avoiding unnecessary transaction boundaries.

There are some general guiding principles at work here:

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

We discuss this in more detail in the following sections.

### 2.5.1 Exploit SCA transaction qualifiers

In an SCA assembly, the number of transaction boundaries can be reduced by allowing transactions to propagate across components. For any pair of components where this is desired, we recommend using the *golden path* (shown in Example 2-1).

*Example 2-1 Golden path*

---

```
SuspendTransaction= false, for the calling component's reference
joinTransaction= true, for the called component's interface
Transaction= any|global, for both components' implementation
```

---

This approach assumes that the first component in such a chain either starts or participates in a global transaction.

## 2.5.2 Use transactional attributes for activities in long-running processes

While SCA qualifiers control component-level transactional behavior, there are additional transactional considerations in long-running business processes that can cause activities to be run in multiple transactions. The scope of those transactions and the number of transactions can be changed with the transactional behavior settings on Java Snippet, Human Task, and Invoke activities. See the WebSphere Process Server Information Center for a detailed description of these settings:

[http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r1mx/topic/com.ibm.websphere.bpc.610.doc/doc/bpc/cprocess\\_transaction\\_macro.html](http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r1mx/topic/com.ibm.websphere.bpc.610.doc/doc/bpc/cprocess_transaction_macro.html)

There are four choices:

- ▶ Commit before
- ▶ Commit after
- ▶ Participates
- ▶ Requires own

Only the participates setting does not require a new transaction boundary. The other three 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 should be used 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 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 may be preferable. Refer to the Information Center link above for details.

Use *commit before* in parallel activities that start new branches to ensure parallelism. As noted in the Information Center, there are other constraints to be considered.

Use *commit after* for inline human tasks to increase responsiveness to human users. When a human user issues a task completion, the thread/transaction handling that action is used to resume navigation of the human task activity in the process flow. The user's task completion action will not complete until the process engine commits the transaction. If the participates setting is used, the commit will get delayed and force a longer response time for the user. This is a classic response time versus throughput trade-off.

Note that starting with the 6.2.0 release, *receive* and *pick* activities in the BPEL flow are now allowed to define their own transactional behavior property values. If not set, the default value of initiating a receive or pick activity is *commit after*. Consider using participates where possible, since participates will perform better.

## 2.6 Invocation style considerations

This section discusses invocation style considerations.

## 2.6.1 Use asynchrony judiciously

Components and modules may be wired to each other either synchronously or asynchronously. The choice of interaction style can have a profound impact on performance and care should be exercised when making this choice.

## 2.6.2 Set the preferred interaction style to sync whenever possible

Many WebSphere Process Server component types like interface maps or business rules invoke their target components based on the target interface's setting of preferred interaction style. Since synchronous cross-component invocations are better performing, we recommend setting 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 this be set to async.

In WebSphere Integration Developer 6.2 when a new component is added to an assembly diagram, its preferred interaction style is set to synchronous, asynchronous, or any based on the component. In previous releases, the default initial setting of preferred interaction style is set to any unless explicitly changed by the user. If a component's preferred interaction style is set to any, how the component is invoked is determined by the caller's context. If the caller is a long-running business 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:

[http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r1mx/topic/com.ibm.websphere.bpc.610.doc/doc/bpc/cprocess\\_transaction.html](http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r1mx/topic/com.ibm.websphere.bpc.610.doc/doc/bpc/cprocess_transaction.html)

Some additional considerations are listed below:

- ▶ When setting an interface's preferred interaction style to async, it is important to realize the downstream implications. Any components invoked downstream will inherit the async interaction style unless they explicitly set preferred interaction style to sync.
- ▶ At the input boundary to a module, exports that represent asynchronous transports like MQ, JMS, or JCA (with async delivery set) will set the interaction style to async. This can cause downstream invocations to be async if the preferred interaction style is left as any.
- ▶ For an SCA import, its preferred interaction style can be used to specify whether the cross-module call should be sync or async.
- ▶ For other imports that represent asynchronous transports like MQ or JMS, it is not necessary to set the preferred interaction style to async. Doing so will introduce an unnecessary async hop between the calling module and the invocation of the transport.

## 2.6.3 Minimize 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. Hence, an asynchronous invocation should be thought of as a full messaging *hop* to its target. When the intended target of the invocation is in the same module, a synchronous invocation will yield much higher performance.

## 2.7 Messaging considerations

This section discusses messaging considerations.

### 2.7.1 Choose MQ or MQ/JMS binding rather than MQ Link

Prior to the 6.0.2 release, the MQ Link product was required in order to directly use MQ queues in conjunction with WebSphere Process Server and WebSphere ESB. However, the MQ and MQ/JMS bindings (added in the 6.0.2 release) are easier to configure and deliver much better performance.

### 2.7.2 Set MaxSession for the MQ/JMS export binding

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

### 2.7.3 Use mediations that benefit from WebSphere ESB optimizations

Certain types of mediation benefit from internal optimization in WebSphere ESB and deliver improved performance. This specialized optimization can be regarded as a kind of fastpath through the code and is in addition to any general optimization of the WebSphere ESB mediation code. The optimization is known as deferred parsing. As the name implies, parsing the message can be deferred until absolutely required, and in several cases (described below) parsing can be avoided altogether.

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

- ▶ Category 1 (greatest benefit)
  - Route on Message Header (Message Filter Primitive)
  - XSLT Primitive (Transforming on /body as the root)
  - EndpointLookup without Xpath user properties.
  - Event Emitter (CBE Header Only)
- ▶ Category 2 (medium benefit)
  - Route on Message Body (Message Filter Primitive)
- ▶ Category 3 (lowest benefit)
  - Custom Mediation
  - Database Lookup
  - Message Element Setter
  - BO Mapper
  - Fan Out
  - Fan In
  - Set Message Type
  - Message Logger
  - Event Emitter (Except for CBE Header only)
  - EndpointLookup 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 above 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 the above 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 the XSLT primitive should be placed ahead of the other primitives. So:

→ **Route On Header** → **XSLT Primitive(/body)** → **Custom Primitive** →

is preferable to:

→ **Route On Header** → **Custom Primitive** → **XSLT Primitive(/body)** →

It should be understood that there are costs 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 mediation will contain code that may not be optimally written. The above guidelines can help in designing for performance, but they cannot guarantee speed.

## 2.7.4 Usage of XSLTs versus BO maps

In a mediation flow that is eligible for deferred parsing, the XSL Transform primitive provides better performance than the Business Object Map primitive. However, in a mediation flow where the message is already being parsed the Business Object Map primitive provides better performance than the XSL Transform primitive.

## 2.7.5 Configure WebSphere ESB resources

When creating resources using WebSphere Integration Developer, 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.

- ▶ Pre-configured resources support:
  - Existing resources to be used
  - External creation/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 use 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 once the applications have been 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 third-party JMS provider software, the tooling-created resources are used to locate the externally defined resources.

## **2.8 Large object best practices**

This section discusses large object best practices.

### **2.8.1 Avoid lazy cleanup of resources**

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

### **2.8.2 Avoid tracing when processing large BOs**

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

Also, avoid constructing trace messages outside of conditional guard statement. For example, the sample code below will create a large string object even if tracing is disabled:

```
String boTrace = bo.toString();
```

While this pattern is always inefficient, it hurts performance even more if the BO size is large. To avoid unnecessarily creating a BO when tracing is disabled, move the string construction inside an if statement, as is shown below

```
if (tracing_on) System.out.println(bo.toString());
```

### **2.8.3 Avoid buffer-doubling code**

Study the memory implications of using Java data structures that 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.

### **2.8.4 Make use of deferred-parsing friendly mediations for XML docs**

Certain mediations can reduce memory pressure as they retain the publication in their native form and avoid inflating them into their full BO representation. These mediations are listed in 2.7.3, “Use mediations that benefit from WebSphere ESB optimizations” on page 22. Where possible, use these mediations.

## 2.9 WebSphere InterChange Server migration considerations

The following list contains useful information for those migrating from WebSphere Interchange™ Server to WebSphere Process Server:

- ▶ Utilize JCA adapters to replace WBI adapters, where possible. Migrated workloads making use of WBI adapters result in interaction with the WebSphere Process Server server through JMS, which is slower than the JCA adapters.
- ▶ Some technology adapters like HTTP and Web services are migrated by the WebSphere InterChange Server migration wizard into native WebSphere Process Server SCA binding, which is a better performing approach. For adapters that are not migrated automatically to available SCA bindings, development effort spent to manually migrate to a SCA binding will remove the dependency on a WBI adapter as well as have better performance.
- ▶ WebSphere InterChange Server Collaborations are migrated into WebSphere Process Server BPEL processes. Examine the resultant BPEL processes for inefficiency. Development effort spent cleaning up the migrated BPEL will result in better performance and better maintainability.
- ▶ Investigate the possibility of replacing BPEL processes produced by migration with other artifacts. All WebSphere InterChange Server collaborations currently get migrated into BPEL processes. For certain scenarios other WebSphere Process Server artifacts may be better choices (for example, business rules). Investigate the BPEL processes produced by migration to ensure that the processes are the best fit for your scenario.
- ▶ Reduce memory pressure by splitting the shared library generated by the migration wizard. The migration wizard creates a single shared library and puts all migrated business objects, maps, and relationships in it. This library is then shared by copy by all the migrated modules. This can cause memory bloat in cases where the shared library is very large and a large number of modules are present. The solution is to manually re-factor the shared library into multiple libraries based on functionality or usage and modify modules only to reference the shared libraries that it needs.
- ▶ Follow the technote *Improving performance of migrated InterChange Server submap calls with native WebSphere Process Server submap calls* to replace custom submap invokes generated by the migration wizard with WebSphere Process Server native submap invokes to improve performance. The technote is available at:

<http://www.ibm.com/support/docview.wss?uid=swg21328875>

## 2.10 Adapters: Configure synchronous event delivery

By default, event delivery 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, we recommend that synchronous semantics be used 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 perform the following steps:

1. Change to the Business Integration perspective and open the appropriate module using the assembly editor. Right-click the adapter export box and select **Show in Properties** → **Binding** → **Performance attributes**. Change the pull-down value in the Interaction Style box to sync.
2. Save your changes, rebuild your application, export your ear, and redeploy.

## 2.11 WebSphere Integration Developer considerations

This section describes recommendations intended to improve the performance of activities commonly encountered by application developers during the development of an enterprise application, primarily import, build and publish of an application workspace.

### 2.11.1 Leverage hardware advantages

Importing and building an enterprise application is, in itself a resource-intensive activity. Recent improvements in desktop hardware architecture have greatly improved the responsiveness of import and build activities. In particular, Intel® Core2 Duo cores perform much better than the older PentiumD architecture, even when the Core2 Duo runs at a slower clock rate. Also, for I/O-intensive activities (like import), a faster disk drive reduces total response time.

### 2.11.2 When appropriate, tune the server for fast publish

At times during the development process (for instance, during debugging) it is advantageous to be able to deploy an application to the server test environment quickly. This section discusses several tuning changes that improve publish responsiveness. Take advantage of these when they are applicable.

#### **Use gencon garbage collection policy on the server**

Generational, concurrent garbage collection, available as of Java5 IBM JVMs (in WebSphere Process Server Version 6.1.0 and later) generally provides improved garbage collection throughput and can be used without reservation.

#### **Use RMI for server communication**

RMI provides a lighter-weight protocol for communication between the development environment (WebSphere Integration Developer) and server, improving publish responsiveness.

#### **Deploy with resources in the workspace and minimize file copies**

These two options, taken together, greatly reduce the amount of data copied from the workspace to the server runtime and provide a dramatic boost in publish responsiveness. These options are available when deploying to a local server (installed on the same physical machine as the development environment).

### 2.11.3 Make use of shared libraries in order to reduce memory consumption

For applications containing many projects utilizing a WebSphere Process Server shared library, server memory consumption is reduced by defining the library as a shared library, as described in the technote found at:

<http://www.ibm.com/support/docview.wss?uid=swg21298478>

## 2.12 Fabric considerations

This section discusses considerations for WebSphere Business Services Fabric (Fabric).



### **2.12.1 Only specify pertinent context properties in context specifications**

The effectiveness of Fabric's runtime caching of metadata is governed by the number of context properties explicitly listed in a context specification. Thus, care should be taken to limit cached content by using only the context properties that are pertinent to a particular dynamic decision. For example, if a credit score context property is not used in a particular dynamic decision, then do not list that context property in the associated context specification.

Note that this applies to strict context specifications, which is the preferred mechanism.

### **2.12.2 Bound the range of values for context keys**

The possible values of a context key should be bound to either a finite set or a minimum and maximum value. The Fabric runtime caches metadata based on the contexts defined as required or optional in the context specification. Thus, having a context key that can take an unbounded integer as its value will result in too many potential cache entries, which will make the cache less efficient. Consider using classes of possible values rather than absolute numbers. For example, for credit scores group the possible values under poor, average, good, and excellent, rather than using the actual values. The actual values will be placed in one of these categories and that category is then passed as the context.





## Performance tuning and configuration

In order to optimize performance it is usually necessary to configure the system differently than the default settings. This chapter lists several areas to consider during system tuning. This includes tuning the 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 would likely offer the best guidance for performance considerations in a variety of operational environments. Assuming that all these issues have been addressed from the perspective of the actual product, additional levels of performance implications are introduced at the interface between these products and the products covered in this IBM Redpaper.

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.

The next section describes a methodology to use when tuning a deployed system. It is followed by 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.

**Note:** While there is no guarantee that following the guidance in this chapter will immediately provide acceptable performance, it is likely that degraded performance can be expected if these parameters are incorrectly set.

Finally, the last section of this publication contains “Related publications” on page 71 that may 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 we do not exhaustively describe the art of system performance tuning here, which requires training and experience. Rather, we highlight 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 parameters tuning
- ▶ WebSphere Process Server, WebSphere Application Server, and ME tuning

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

1. Pick a set of reasonable initial parameter settings and run the system.
2. Monitor the system to obtain metrics that indicate whether performance is being limited.
3. Use monitoring data to guide further tuning changes.
4. Repeat until done.

We now examine each in turn:

1. Choose a set of reasonable initial parameter settings:
  - a. Use the tuning checklist in 3.2, “Tuning checklist” on page 31, for a systematic way to set parameters.
  - b. For specific initial values, consult Chapter 4, “Initial configuration settings” on page 63, for settings that were used for representative workloads that were evaluated by the IBM performance team. These values can be considered for initial values.
2. Monitor the system. We recommend monitoring the systems to determine system health, as well as to determine the need for further tuning. The following should be monitored:
  - For each physical machine in the topology including front-end and back-end servers like Web servers, and DB servers, monitor processor core utilization, memory utilization, disk utilization, and network utilization using relevant OS tools like vmstat, iostat, netstat, or equivalent.
  - For each JVM process started on a physical machine (that is, WebSphere Process Server server, ME server, and so on):
    - Use tools like “ps” or the equivalent to get processor core and memory usage per process.
    - Collect verbosegc statistics.
  - For each WebSphere Process Server or ME JVM, use Tivoli® Performance Viewer (TPV) to monitor the following:
    - For each data source, the data connection pool utilization
    - For each thread pool (Web container, default, work managers), the thread pool utilization
3. Use monitoring data to guide further tuning changes

This 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 collected in the previous phase.

Examples of performance bottlenecks include, but are not limited to:

- Excessive utilization of physical resources like processor cores, disk, memory, and so on. These can be resolved 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 on. For these, tuning parameters should be used to remove the bottlenecks.

## 3.2 Tuning checklist

This checklist serves as a guide or *to do* list when tuning a WebSphere BPM solution. Each of these topics is covered 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.
- Do not enable security, where practical. If security is required use application security, not Java2 security.
- Use appropriate hardware configuration for performance measurement, for example, ThinkPads and desktops are not appropriate for realistic performance evaluations.
- If hardware virtualization is used, ensure that adequate processor, memory, and I/O resources are allocated to each virtual machine. Avoid over-committing resources.
- Do not run the production server in development mode or with a development profile.
- Do not use the Unit Test Environment (UTE) for performance measurement.
- Tune external service providers and external interfaces to ensure that they are not the system bottleneck.
- Configure MDB Activation Specs.
- Configure for clustering (where applicable).
- Configure thread pool sizes.
- Configure data sources: Connection Pool size, prepared statement cache size. Consider using non-XA data sources for CEI data when that data is non-critical.

### ► Business Process Choreographer

- Use work-manager-based navigation for long-running processes.
- Optimize Business Flow Manager resources:
  - Database connection (BPEDB)
  - Activation specification (BPEInternalActivationSpec)
  - JMS connection (BPECF and BPECFC).
- If work-manager-based navigation is used, also optimize message pool size and intertransaction cache size.
- Optimize the database configuration for the Business Process Choreographer database (BPEDB).
- Optimize indexes for SQL statements that result from task and process list queries using database tools like the DB2 design advisor.
- Turn off state observers that are not needed (for example, turn off audit logging).

- ▶ Messaging and message bindings
  - Optimize activation specification (JMS).
  - Optimize queue connection factory (JMS, MQJMS, MQ).
  - Configure connection pool size (JMS, MQJMS, MQ).
  - Optimize listener port configuration (MQJMS, MQ).
  - Configure SIBus data buffer sizes.
- ▶ Database
  - Place database tablespaces and logs on a fast disk subsystem.
  - Place logs on separate device from tablespace containers.
  - Maintain current indexes on tables.
  - Update database statistics.
  - Set log file sizes correctly.
  - Optimize 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.
  - 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 IBM Redpaper. 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, all of these parameters can be set via 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. Minimize their use when evaluating performance or in production environments.

To enable or disable tracing, go to **Troubleshooting** → **Logs and Trace** → **server name**. Change the log detail levels for 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**.

In addition, Cross-Component Tracing (XCT) is a new capability for logging and tracing, enabling correlation of Service Component Architecture (SCA) component information with log entries. There are two levels of XCT settings:

- ▶ Enable.
- ▶ Enable with data snapshot.

Both incur significant performance overhead. Enable with data snapshot is particularly costly because of the additional I/O involved in saving snapshots in files.

To enable or disable Cross-Component Trace, go to **Troubleshooting** → **Cross-Component Trace**. Select the XCT setting from three options, disable, enable, or enable with data snapshot, in configuration or runtime, or both. Changes to runtime take effect immediately while changes to configuration require a server restart to take effect.

### 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 offered by the JVM supplier.

The JVM admin panel can be accessed from **Servers** → **Application Servers** → *server name* → **Server Infrastructure** → **Java and Process Management** → **Process Definition** → **Additional Properties** → **Java Virtual Machine**.

#### Java GC policy

The default garbage collection algorithm on platforms with an IBM JVM is mark-sweep-compact. In many 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 admin panel.

#### Java heap sizes

To change the default Java heap sizes, set the initial heap size and maximum heap size explicitly on the Java Virtual Machine admin 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 one fourth and one half 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 ActivationSpec

There are a few shortcuts to access the MDB ActivationSpec tuning parameters:

- ▶ **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 ActivationSpec have considerable performance implications. These are discussed further in “Tune MDB ActivationSpec properties” on page 38. The two custom properties are in the MDB ActivationSpec with considerable performance implications:

- ▶ `maxConcurrency`
- ▶ `maxBatchSize`

### 3.3.4 MQ listener port

For the MQ or MQJMS binding, a listener port is used for configuring the delivery of inbound messages to WebSphere Process Server or WebSphere ESB. The maximum sessions property of the listener port has a 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. The maximum size property of a thread pool can be set under **Servers** → **Application servers** → **server name** → **Additional Properties** → **Thread Pools** → **thread pool name**.

The following thread pools typically must be tuned:

- ▶ `Default`
- ▶ `ORB.thread.pool`
- ▶ `WebContainer`

In addition, thread pools used by work managers are configured separately via **Resources** → **Asynchronous beans** → **Work managers** → *work manager name* → **Thread pool properties**.

The following work managers typically must be tuned:

- ▶ `DefaultWorkManager`
- ▶ `BPENavigationWorkManager`



### 3.3.6 JMS connection pool sizes

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

- ▶ **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 admin panel, open **Additional Properties** → **Connection pool properties**. Set the maximum connections property for the max size of the connection pool.

### 3.3.7 Data source connection pool size

Data sources can be accessed 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 can be configured from the data source panel's **Additional Properties** → **Connection pool properties**.

The following data sources typically must be tuned:

- ▶ BPEDataSource for the BPE database
- ▶ SCA Application Bus ME data source
- ▶ SCA System Bus ME data source
- ▶ CEI Bus ME data source

### 3.3.8 Data source prepared statement cache size

The data source prepared statement cache size can be configured from the data source's **Additional properties** → **WebSphere Application Server data source properties**.

The Business Process Choreographer (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 Utilize non-XA data sources for CEI data, if possible

Non-XA data sources can be used for CEI when the CEI data is not critical to the enterprise. They must be created from the data sources list.

### 3.3.10 Messaging engine properties

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

- ▶ DiscardableDataBufferSize
- ▶ CachedDataBufferSize

The properties can be accessed under **Service Integration** → **Buses** → *bus name* → **Messaging Engines** → *messaging engine name* → **Additional properties** → **Custom properties**.

### 3.3.11 Run production servers in production

WebSphere application servers can be run in development mode, which may reduce startup time for the server by using JVM settings to disable bytecode verification and reduce JIT compilation time. This setting should not be used on production servers, however, since it is not designed to produce optimal runtime performance. Make sure that the Run in development mode check box on the administrative console panel **Servers** → **Application Servers** → *server name* → **Configuration** is unchecked.

Server profiles may also be created with production or development templates. Use production profile templates for production servers.

## 3.4 Advanced tuning

This section contains advanced tuning tips.

### 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 via the Common Event Interface and audit logging, and system performance monitoring via the Performance Monitoring Infrastructure (PMI) and the Application Response Measurement (ARM) infrastructure. While these capabilities provide insight into the performance of the running solution, these features can degrade overall system performance and throughput.

**Note:** We recommend that tracing and monitoring be used judiciously and, when possible, turned off entirely to ensure optimal performance.

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

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

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

### 3.4.2 Tuning for large objects

In this section we discuss tuning for large objects.

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

One of the key factors affecting large object processing is the maximum size of the Java heap. In this section we discuss how to set the heap size as large as possible on two commonly used platforms. For more comprehensive heap setting techniques, consult 3.4.13, “Advanced Java heap tuning” on page 56.

### ► 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 JVM supports heaps 2 GB to 2.4 GB on 32-bit systems. Note that since 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 on. An extremely large heap squeezes address space reserved for other resources and may 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 must be made when considering large objects is that not all objects will be *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 being processed concurrently the likelihood of failure increases dramatically.

The size and number of the *normally arriving* smaller objects will 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 poll period and poll quantity” on page 47.

Additionally, for the mySAP™.com® adapter when using the ALE module for event delivery set the connector-specific NumberOfListeners configuration property to 1. This will limit 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 will be many operations that take place simultaneously. Tuning for maximum concurrency ensures that the server will accept enough load to saturate its processor cores. One sign of an inadequately tuned configuration is when additional load does not result in additional processor core utilization, while the processor cores 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 must be balanced with other tuning objectives, such as the handling of large objects, handling large numbers of users, and providing good response time.

### **Tune edge components for concurrency**

The first step is to ensure that business objects are handled concurrently at the edge components of WebSphere Process Server or WebSphere ESB. If the input BOs come from the adapter, ensure that the adapter is tuned for concurrent delivery of input messages. See 3.4.8, “WebSphere adapters tuning” on page 47, for more details on tuning adapters.

If the input BOs come from WebServices export binding or direct invocation from JSP™ or Servlets, make sure that the WebContainer thread pool is correctly sized. To allow for 100 in-flight requests handled concurrently by the WebSphere Process Server, the maximum size of the WebContainer thread pool must be set to 100 or larger.

If the input BOs come from the messaging, the ActivationSpec (MDB bindings) and Listener ports (MQ or MQJMS bindings) must be tuned.

### **Tune MDB ActivationSpec properties**

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

Note that the Tivoli Performance Viewer (TPV) can be used to monitor the maxConcurrency parameter. For each message being processed by an MDB there will be a message on the queue marked as being locked inside a transaction (which will be removed once the onMessage completes). These messages are classed as unavailable. There is a PMI metric that gives you the number of unavailable messages on each queue point (**resource\_name** → **SIB Service** → **SIB Messaging Engines** → **bus\_name** → **Destinations** → **Queues**), which is called UnavailableMessageCount. If any queue has at least maxConcurrency unavailable messages it implies that it is currently running above the MDB's concurrency maximum. If this occurs, increase the maxConcurrency setting for that MDB.

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.) Increase this value (for example, to 8) for activation specs associated with SCA modules and long-running business processes to improve performance and scalability, especially for large multi-core systems.

### **Configure 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 ActivationSpec for JMS bindings. Increase this value to an appropriate value.

Note that a listener port will be created when the SCA module containing the MQ or MQ/JMS binding is deployed 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 must be set to optimal values. Increasing the `maxConcurrency` or maximum sessions parameters only enables the concurrent delivery of BOs 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 must 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 must 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 must be large enough to handle requests coming in through the 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 the steps below to change the thread pool used for JMS MDB threads:

1. Create a new thread pool, such as `MDBThreadPool`, on the server.
2. Open the Service Integration Bus (SIB) JMS Resource Adapter admin panel with server scope from **Resources** → **Resource Adapters** → **Resource adapters**. If the adapter is not shown, go to **Preferences** and select the **Show built-in resources** check box.
3. Change the thread pool alias from the 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 take effect.

SCA Module MDBs for asynchronous SCA calls use a separate resource adapter, the Platform Messaging Component SPI Resource Adapter. Follow the same step as above 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 BO 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

ActivationSpec'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 (for more details go to 2.5, "Transactionality considerations" on page 19). BPE defines an internal MDB and its ActivationSpec: BPEInternalActivationSpec. The maxConcurrency parameter must be tuned following the same guideline as for a SCA module and JMS export MDBs (described above). The only catch is there is one BPEInternalActivationSpec in the WebSphere Process Server server.

## Configure JMS and JMS queue connection factories

Multiple concurrently running threads may 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 backend 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 resources 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 maximum connections is set to 0, the connection pool is allowed to grow infinitely. This also has the side effect of causing the connection timeout value to be ignored.

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

For each JMS, MQ, or MQJMS import, there is a Connection Factory created during application deployment. 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.

From the Connection Factory admin panel, open **Additional Properties** → **Connection pool properties**. Set the maximum connections property to the max size of the connection pool

## 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/WebSphere ESB servers (for example, BPEDB data source, WPSDB data source, and Message Engine DB data sources). Set each data source's maximum connection property to match the maximum concurrency of other system resources as discussed previously in this chapter.

## 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 provides tuning recommendations for messaging.

#### For message engines, choose datastore or filestore

Message engine persistence is usually backed by a database. For a standalone configuration of WebSphere Process Server or WebSphere ESB V6.2.0, the persistence storage of BPE and SCA buses can be backed by the file system (filestore). The choice of filestore must be made at profile creation time. Use the profile management tool to create a new standalone enterprise service bus profile or standalone process server profile. Choose **Profile Creation Options** → **Advanced profile creation** → **Database Configuration** and select the **Use a file store for Messaging Engine (MEs)** check box. When this profile is used, filestores will be used 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, since this data is never written to the data store for this quality of service. Messages that 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.

The `DiscardableDataBufferSize` and `CachedDataBufferSize` can be set under **Service Integration-Buses** → *bus name* → **Messaging Engines** → *messaging engine name* → **Additional properties** → **Custom properties**.

#### Move message engine datastores to a high-performance DBMS

For better performance, the message engine datastores should use production-quality databases, such as DB2, rather than the default Derby. The choice can be made at profile creation time using advanced profile creation option. If the profile has already been created with Derby as the ME datastore, the following method can be used to change the datastore to an alternative database.

After the Profile Creation Wizard 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 machine box01. The node and cell names are the default.

Table 3-1 Buses in WebSphere Process Server installed on machine box01

Bus	Messaging engine
SCA.SYSTEM.box01Node01Cell.Bus	box01-server1.SCA.SYSTEM.box01Node01Cell.Bus
SCA.APPLICATION.box01Node01Cell.Bus	box01-server1.SCA.APPLICATION.box01Node01Cell.Bus
CommonEventInfrastructure_Bus	box01-server1.CommonEventInfrastructure_Bus
BPC.box01Node01Cell.Bus	box01-server1.BPC.box01Node01Cell.Bus

Each of these message engines is by default configured to use a datastore in Derby. Each datastore is located in its own database. For DB2, this 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 proposed here uses a single DB2 database for all four datastores. The individual datastores/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 datastores see "Messaging engines and datastores" in the WebSphere Application Server V6 Information Center.

### Create the DB2 database and load the datastore 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. See Table 3-2.

Table 3-2 Loading datastore schemas

Schema	Messaging engine
SCASYS	box01-server1.SCA.SYSTEM.box01Node01Cell.Bus
SCAAPP	box01-server1.SCA.APPLICATION.box01Node01Cell.Bus
CEIMSG	box01-server1.CommonEventInfrastructure_Bus
BPCMSG	box01-server1.BPC.box01Node01Cell.Bus

Follow these steps:

1. Create one schema definition for each messaging engine with the following command. (The command shown below uses Windows semantics.)

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

<WAS Install> represents the WebSphere Process Server installation directory, <user> represents the user name, and <path> represents the fully qualified path to the referenced file.

2. Repeat this command for each schema/messaging engine.

To be able 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 like Example 3-1 after editing.

Example 3-1 Schema definition

```
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 tablespaces for the various tables here. Optimal distribution depends on application structure and load characteristics. In this example one table space per datastore was used.

3. After creating all schema definitions and defined tablespaces for the tables, create a database named SIB.
4. Create the tablespaces 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' )
```

5. Place the database log on a separate disk if possible.
6. Create the schema of the database by loading the four schema definitions into the database.

See 3.4.10, “Database: general tuning” on page 51 and 3.4.11, “Database: DB2-specific tuning” on page 51 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 datastore using the administrative console. Table 3-3 shows the default states.

Table 3-3 Messaging engine default states

Messaging engine	JDBC provider
box01-server1.SCA.SYSTEM.box01Node01Cell.Bus	Derby JDBC Provider (XA)
box01-server1.SCA.APPLICATION. box01Node01Cell.Bus	Derby JDBC Provider
box01-server1.CommonEventInfrastructure_Bus	Derby JDBC Provider
box01-server1.BPC.box01Node01Cell.Bus	Derby JDBC Provider

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, and the remaining three as single-phase commit (non-XA) data sources. Table 3-4 provides new names.

Table 3-4 New data source names

Name of data source	JNDI name	Type of JDBC provider
CEIMSG_sib	jdbc/sib/CEIMSG	DB2 Universal (XA)
SCAAPP_sib	jdbc/sib/SCAAPPLICATION	DB2 Universal
SCASYSTEM_sib	jdbc/sib/SCASYSTEM	DB2 Universal
BPCMSG_sib	jdbc/sib/BPCMSG	DB2 Universal

When creating a data source:

1. Uncheck the Use this Data Source in container managed persistence (CMP) check box.
2. Set a component-managed authentication alias.
3. Set the database name to the name used for the database created earlier for messaging, for example, SIB.
4. Select a driver type of 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 datastores of the messaging engines***

Use the administrative console to change the datastores of the messaging engines:

1. In the Navigation panel go to **Service Integration** → **Buses** and change the datastores for each Bus/Messaging Engine displayed.
2. Enter the new JNDI and schema name for each datastore. Uncheck the Create Tables check box since the tables have been created already.

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

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

The last remaining task is tuning the database. See 3.4.10, “Database: general tuning” on page 51 and 3.4.11, “Database: DB2-specific tuning” on page 51 for further information about database and DB2-specific tuning, respectively.

## **3.4.5 Web services tuning**

If the target of the Web services 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 Web services clients are sent through the network connection between the client and the service provider. For local Web services calls, however, WebSphere Application Server offers a direct communication channel, bypassing the network layer completely. Follow the steps below to enable this optimization. Use the WebSphere administrative console to make these changes.

1. Set Web container custom property enableInProcessConnections to true at **Application servers** → **server name** → **Container Settings** → **Web Container Settings** → **Web container** → **Additional Properties** → **Custom Properties**.
2. Do not use a wildcard (\*) for the host name of the Web container port. Replace it with the host name or IP address. The property can be accessed from **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**.
3. Use localhost instead of host name in the Web services client binding. If the actual host name is used and even if it is aliased to localhost, this optimization will be disabled. The property can be accessed from **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.
4. Make sure that 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.

### 3.4.6 Business Process Choreographer tuning

This section provides advanced tuning tips for the Business Process Choreographer.

#### Tuning the business process container

The following resources must be optimized for efficient navigation of business processes. Particularly for JMS navigation, the following parameters have an impact on navigation performance:

- ▶ Activation specification `BPEInternalActivationSpec`: The maximum concurrent endpoints parameter specifies the parallelism that is used for process navigation across all process instances. Increase the value of this parameter to increase the number of business processes executed concurrently. This resource can be found at **Resources** → **Activation Specifications** → **BPEInternalActivationSpec**.
- ▶ JMS connection factory `BPECFC`: This connection factory is used when JMS-based navigation is used. If work-manager-based navigation is used, this connection factory is only used in error situations in case the server is very overloaded. If JMS-based navigation is used, set the connection pool size to the number of threads of the `BPEInternalActivationSpec` + 10%. This resource can be found at **Resources** → **JMS** → **Connection factories** → **BPECFC** → **Connection pool properties**.

#### Tuning task list and process list queries

Task list and process list queries in Business Process Choreographer applications are made using the standard query API (`query()` and `queryAll()` APIs, and related REST and Web services interfaces) and the query table API (`queryEntities()` and `queryRows()` APIs). All task list and process list queries result in SQL queries against the Business Process Choreographer database. These SQL queries might need special tuning in order to provide optimal response times:

- ▶ Up-to-date database statistics are key for good SQL query response times.
- ▶ Databases offer tools to tune SQL queries. In most cases, additional indexes are very helpful. For DB2, the DB2 design advisor can be used.

#### Tuning Business Process Choreographer API calls

Business Process Choreographer API calls are triggered by requests external to the WebSphere Process Server runtime. Examples are remote EJB requests, Web service requests, Web requests over HTTP, requests that come through the SCA layer, and JMS requests. The connection pools associated with each of these communication mechanisms may need to be tuned. Consider the following hints when tuning the connection pools:

- ▶ Business Process Choreographer API calls have different response time characteristics from other messaging applications. API calls for task list and process list queries may take more time to receive a response, depending on the tuning of the database and the amount of data in the database.
- ▶ Ensure that concurrency (parallelism) is sufficiently high to handle the offered load, but not excessively high. Increasing the parallelism of API call execution beyond what is necessary may negatively influence response times. Also, increased parallelism may put excessive load on the database (BPEDB). When tuning the parallelism of API calls, measure response times before and after tuning, and adjust the parallelism if necessary.

#### Tuning WorkManager-based navigation for business processes

WebSphere Process Server 6.1.0.1 introduced a new navigation mode, WorkManager-based navigation, as opposed to the default JMS based navigation. In WebSphere Process Server 6.2.0, the capability is enhanced for easier use.

WorkManager-based navigation enables two performance optimizations:

- ▶ WorkManager-based navigation: A WorkManager is a thread pool for J2EE threads. The use of a WorkManager allows the BPC engine to exploit an underlying capability of WebSphere Application Server to start the processing of ready-to-navigate business flow activities without using messaging as provided by JMS providers.
- ▶ The interTransactionCache, which is a part of the new navigation mode that holds process instance state information in memory, reducing the need to retrieve information from the BPE database.

There are several parameters that control usage of these two optimizations. The first set of these parameters is found by going to **Application Servers** → **server name** → **Business Integration** → **Business Process Choreographer** → **Business Flow Manager** → **Business Process Navigation Performance**.

The key parameters are:

- ▶ Check **Enable advanced performance optimization** to enable both the WorkManager-based navigation and interTransactionCache optimizations.
- ▶ WorkManager Based Navigation Message Pool Size: This property specifies the size of the cache used for BPE Navigation messages that cannot be processed immediately, provided that WorkManager-based navigation has been enabled. The cache defaults to a size of (10 \* thread pool size of the BPENavigationWorkManager) messages. Note that if this cache reaches its limit, the BPC engine uses JMS-based navigation for new messages.
- ▶ InterTransaction Cache Size: This property specifies the size of the cache used to store process state information that has also been written to the BPE database. It should be set to twice the number of parallel running process instances. The default value for this property is the thread pool size of the BPENavigationWorkManager.

In addition, customize the number of threads for the work manager via **Resources** → **Asynchronous Beans** → **Work Managers** → **BPENavigationWorkManager**.

The minimum and maximum number of threads should be increased from their default values of 5 and 12, respectively, using the methodology outlined in 3.4.3, “Tuning for maximum concurrency” on page 37. If thread pool size is modified, then the work request queue size should also be modified and set to be twice the maximum number of threads.

### 3.4.7 WebSphere ESB tuning

Following are additional configuration options that are relevant to tuning WebSphere ESB. See 4.2, “WebSphere ESB settings” on page 68 for a representative list of the values that can be used to tune a WebSphere ESB configuration.

#### 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, such as the DB server. You may also benefit from tuning the connection pooling and statement cache of the data source. See 3.4.10, “Database: general tuning” on page 51 and 3.4.11, “Database: DB2-specific tuning” on page 51 for further information about tuning DB2, and also note the relevant publications in “Related publications” on page 71.

## Disable event distribution for CEI

The event server that manages events can be configured to distribute events and 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. Since the event server may be used by other applications it is important to check that none of them use event monitoring that requires event distribution before disabling this.

Event distribution can be disabled from **Service integration** → **Common Event Infrastructure** → **Event service** → **Event services** → **Default Common Event Infrastructure event server**. Uncheck Enable event distribution.

## Configure WSRR cache timeout

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

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

## 3.4.8 WebSphere adapters tuning

In this section we provide the configuration options that are relevant to tuning WebSphere Adapter applications.

### Configure poll period and poll quantity

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

- ▶ pollPeriod specifies the amount of time (in milliseconds) between polling actions.
- ▶ pollQuantity specifies the maximum number of events to process during a polling action.

Since these parameters control the rate and amount of work that an adapter will process, 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 poll period is too short or poll quantity is too high). Since both of these conditions dramatically impact overall system performance, appropriate settings for poll period and poll quantity are critical, and should be explicitly configured to support the level of throughput that a solution is designed to handle.

There are two possible scenarios that must be kept in mind. In the first one 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 out-of-memory conditions.

In general, we recommend configuring `pollPeriod` and `pollQuantity` to enable events to be retrieved and processed at a level that matches the peak throughput of the overall solution. We discuss this 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 poll period enables a minimal delay between poll cycles. Factors that may require an adjustment to these values include:

- ▶ The size of the object being processed

For larger objects, a good rule of thumb is to use a lower `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 two 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 that it takes to process the `pollQuantity` events (two in this case) and adjusting the poll period for that.

- ▶ The Java heap size and physical memory 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 will dramatically reduce system performance. See 3.4.13, “Advanced Java heap tuning” on page 56 for a detailed discussion about 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 three times the size of the object, but this varies depending on the adapter and the business process.

- ▶ The uniformity of event arrival rates

The examples above assume that events arrive at a relatively constant rate. This may not be true for many solutions. Event arrival is sometimes very uneven. In these cases, care must be taken 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.

- ▶ Number of objects in-flight

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 will then start preparing other files in the event delivery directory for processing before the first file has completed, possibly leading to an out-of-memory 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 will be constrained, and either `OutOfMemory` exceptions could occur or garbage collection time will be increased, resulting in reduced performance.

### Configure the application server thread pool

For event delivery when using the default thread pool, increasing the number of threads available to the adapter can improve performance. The number of threads should be set to a value that includes the needs of the server and applications, as well as the JCA adapter. To set the number of threads using the WebSphere administrative console go to **Servers** → **Application servers** → **server name** → **Additional Properties** → **Thread Pools** → **Default**. Update the maximum and minimum values, 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 must be tuned. To set this via the WebSphere administrative console go to **Resources** → **Asynchronous beans** → **Work Managers** → **DefaultWorkManager** (or your application-specific work manager) and set the minimum and maximum number of threads. We recommend 20 for both min/max as a starting point, but depending on your configuration you might need higher values.

### Flat files, ftp: Place event delivery and archive directories on same disk

Placing these 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 like flat files, FTP, e-mail, and SAP® can use external event tables. JMS queues may also be used at various points, for example, during asynchronous event delivery. Periodically monitor both the tables and JMS queues to ensure that events are being processed and removed from these queues. Obsolete events can produce delays in processing, since the adapter may attempt to unnecessarily recover these events. External table names can be found by using the WebSphere administrative console. The names are specified in the J2C Activation Spec of the deployed adapter. These tables can then be monitored and cleared via 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 WebSphere administrative console. Again, if using DB2, they are specified as data sources in the Service Integration and Resources section.

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

## 3.4.9 WebSphere Business Monitor tuning

This section provides advanced tuning recommendations for WebSphere Business Monitor.

### 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 Monitor Launchpad installs Monitor and its prerequisite servers with larger heap sizes, but you might check that these sizes are appropriate for your hardware and workload. If you have sufficient physical memory, start by specifying initial and maximum heap sizes of 1536 M and 1536 M, respectively.

## Configure CEI

By default, when an event arrives at CEI, it is delivered to the registered consumer (in this case a particular monitor model) and also into an additional, default queue. Unless the customer needs this extra queue, do not double-store. You can prevent this extra store in the WebSphere administrative console by removing the *all events* event group found via **Service Integration** → **Common Event Infrastructure** → **Event Service** → **Event Services** → **Default Common Event Infrastructure event server** → **Event Groups**.

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 in the WebSphere administrative console by going to **Service Integration** → **Common Event Infrastructure** → **Event Service** → **Event Services** → **Default Common Event Infrastructure event server** and deselecting **Enable Data Store**.

## Configure message consumption batch size

Consuming events 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. But there is a trade-off: Consuming events, processing them, and persisting them to the Monitor database is done as a transaction. So while a larger batch size yields better throughput, it will cost more if you must roll back. By default, WB Monitor sets the batch size to 100 events. If you experience frequent rollbacks, consider reducing the batch size. This can be done in the WebSphere administrative console in the Server Scope by going to **Applications** → **Monitor Models** → *version* → **Runtime Configuration** → **Tuning** → **Message Consumption Batch size**: `<default 100>`.

## Configure WB Monitor work manager maximum threads

The processing of incoming events for a model is done through the use of several WorkManager thread pools. There are two thread pools whose maximum number of threads property may be changed to achieve full use of all available processor cores on a Monitor server. The sizes can be configured by going to **Resources** → **Asynchronous beans** → **Work managers** → *name*. See Table 3-5.

Table 3-5 Names and values

Name	Value
wm_wbm_Moderator_EventConsumption	1
wm_wbm_Moderator_EventParsing	10
wm_wbm_Moderator_EventProcessing	20
wm_wbm_Moderator_EventFragmentInsertion	1
wm_wbm_Moderator/FragmentReadiness	3
wm/wbm/<model>_<version>_FragmentProcessing	1

Also, make sure that **growable** is not selected.

## Enable KPI caching

The cost of calculating aggregate KPI values increases as completed process instances accumulate in the database. A KPI cache is available to reduce the overhead of these calculations, at the cost of some staleness in the results. The refresh interval is configurable by going to **Applications** → **Monitor Models** → *version* → **Runtime Configuration** → **KPI** → **KPI Cache Refresh Interval**. A value of zero (the default) disables the cache.



### 3.4.10 Database: general tuning

This section contains general tuning recommendations for databases.

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

Databases are designed for high availability, transactional processing, and recoverability. Since for performance reasons changes to table data may not 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 may be heavily utilized. 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 database log files be placed on a fast disk subsystem with write back cache.

#### **Place logs on separate device from Tablespace Containers**

A basic strategy for all database storage configurations is to place the database logs on separate physical disk devices from the tablespace containers. This reduces disk access contention between I/O to the tablespace containers and I/O to the database logs and preserves the mostly sequential access pattern for the log stream. Such separation also improves recoverability when log archival is employed.

### 3.4.11 Database: DB2-specific tuning

Providing a comprehensive DB2 tuning guide is beyond the scope of this IBM Redpaper. However, there are a few general rules of thumb that can assist in improving the performance of DB2 environments. In the sections below we discuss these rules, and provide references to more detailed information. The complete set of current DB2 manuals (including database tuning guidelines) can be found by through the DB2 Information Center:

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

#### **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. 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. It may be turned on from the DB2 Control Center. Updating statistics allows the DB2 query optimizer to create better performing access plans for evaluating queries.

One approach to manually updating statistics on all tables in the database is using the REORGCHK command. Dynamic SQL, such as that produced by JDBC, immediately takes the new statistics into account. Static SQL, like that in stored procedures, must be explicitly rebound in the context of the new statistics. Example 3-2 performs these steps on database DBNAME.

*Example 3-2 Steps performed on database DBNAME*

---

```
db2 connect to DBNAME
db2 reorgchk update statistics on table all
db2 connect reset
db2rbind DBNAME all
```

---

The REORGCHK and rebind should be executed when the system is relatively idle so that a stable sample may be acquired and to 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
- ▶ The number of pages allocated

Buffer pool page size is fixed at creation time and may be set to 4, 8, 16, or 32 KB. The most commonly used buffer pool is IBMDEFAULTBP, which has a 4 KB page size. Starting with Version 8, 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 that all buffer pools reside in database global memory, allocated on the database machine. The buffer pools must coexist with other data structures and applications, all without exhausting available memory. In general, having larger buffer pools will 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 may 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 I/O on devices holding the tablespace 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. While 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 logspace is governed by the product of log file size and the number primary log files, which are configured at the database level. logfilsiz is the number of 4 K pages in each log file. logprimary 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 softmax, larger log files mean that recovery may take more time. The softmax parameter can be lowered to counter this, but keep in mind that more aggressive page cleaning may also be less efficient. Increasing softmax provides additional opportunities for write coalescing at the cost of longer recovery time.

## DB2 9.5: Use SMS instead of DMS when using LOBS or LFs

DB2 buffer pools do not cache large field (LF) or large object (LOB) data from tables. The file system cache of an operating system is capable of caching this data and reducing the physical disk activity. When DB2 has file system caching enabled, all data including the I/O activity from the disk to the buffer pools goes through the file system cache. It is not necessary for data that can reside in buffer pools to go through the file system cache in a properly tuned system. In some cases, caching at the file system level and in the buffer pools causes performance degradation because of the extra CPU cycles required for the double caching. Ideally, LF and LOB data should take advantage of the file system cache, but other activity to and from the buffer pools should bypass it.

By default, DB2 9.5 database creation has file system caching off if the file system allows it to avoid double caching. In addition, the user table space for a database defaults to database managed space (DMS), with automated storage enabled. This combination avoids double caching but does not allow for any file system caching of LF or LOB data.

If DB2 tablespaces are created with system managed space (SMS), LF and LOB data utilize the file system cache regardless of the DB2 setting for file system caching. This can have a big impact by reducing disk utilization for LF and LOB data disk activity. This is especially true for the Business Process Choreographer database (BPEDB) and message engine datastores, which use a great number of LOB files in their tables. The following link discusses specifying initial DB2 database settings with examples of creating SMS tablespaces for the

BPEDB. It also contains useful links for planning the BPEDB database and fine-tuning the Business Process Choreographer database:

[http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r2mx/index.jsp?topic=/com.ibm.websphere.bpc.620.doc/doc/bpc/t5tuneint\\_spec\\_init\\_db\\_settings.html](http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r2mx/index.jsp?topic=/com.ibm.websphere.bpc.620.doc/doc/bpc/t5tuneint_spec_init_db_settings.html)

The following link discusses creating a DB2 for Linux®, UNIX®, and Windows database for Business Process Choreographer and provides details on DB2 BPEDB database creation, with pointers to useful creation scripts, for a production environment:

<http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r2mx/index.jsp?topic=/com.ibm.websphere.bpc.620.doc/doc/bpc/t2codbdb.html>

## **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 K pages set aside for this use. A second database level parameter, maxlocks, bounds the percentage of the lock pool 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, application behavior can be altered to reduce pressure on the lock pool by breaking up large transactions that 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, 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
```

Current values for locklist and maxlocks can be obtained by examining the output from:

```
db2 get db config for yourDatabaseName
```

These values can be altered, for example, to 100 and 20, like this:

```
db2 update db config for yourDatabaseName using locklist 100 maxlocks 20
```

When increasing the locklist size, consider the impact of the additional memory allocation required. Often the locklist 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 locklist size alone.

### 3.4.12 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 the sections below, we discuss these rules and provide references to more detailed information. In addition, the following references are useful:

- ▶ Oracle Database 10g Release 2 documentation (includes a performance tuning guide)  
<http://www.oracle.com/technology/documentation/database10gr2.html>
- ▶ Introduction to Oracle database performance tracing  
[http://www.oracle.com/technology/oramag/oracle/04-jan/o14tech\\_perf.html](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 manually updating statistics on all tables in a schema is to use the `dbms_stats` utility (shown in Example 3-3).

*Example 3-3 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

This reference discusses the issue in detail:

[http://download-uk.oracle.com/docs/cd/B19306\\_01/server.102/b14211/memory.htm#i29118](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 may 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 I/O on devices holding the tablespace datafiles. 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.

### 3.4.13 Advanced Java heap tuning

Because the WebSphere BPM product set is written in Java, the performance of the Java Virtual Machine (JVM) has a significant impact on the performance delivered by these products. JVMs externalize multiple tuning parameters that may be used to improve both tooling and runtime performance. The most important of these are related to garbage collection and setting the Java heap size. This section discusses these topics in detail.

Note that the products discussed in this paper utilize IBM JVMs on most platforms (AIX, Linux, Windows, and so on) and the HotSpot JVMs on selected other systems, such as Solaris™. Vendor-specific JVM implementation details and settings will be discussed as appropriate. Also note that all V6.2.0 products in this publication use Java 5, which is much different from Java 1.4.2 used by V6.0.2.x and earlier releases. For brevity, only Java 5 tunings are discussed here.

Refer to the IBM Java 5 Diagnostics Guide at the following link:

<http://publib.boulder.ibm.com/infocenter/javasdk/v5r0/index.jsp>

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

The following references are for Sun™ HotSpot JVM:

- ▶ A useful summary of HotSpot JVM options for Solaris can be found at:  
<http://java.sun.com/docs/hotspot/VMOptions.html>
- ▶ The following URL provides a useful FAQ about the Solaris HotSpot JVM:  
<http://java.sun.com/docs/hotspot/PerformanceFAQ.html#20>
- ▶ For more performance tuning information about Sun's HotSpot JVM see:  
<http://java.sun.com/docs/performance/>

#### Monitoring garbage collection

In order to set the heap correctly, you must first determine how the heap is being used. This is easily done 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 by using the Java run-time option verbose:gc. Output from verbosegc is different for the HotSpot and IBM JVMs, as shown by the following examples.

Example 3-4 shows IBM JVM verbosegc trace output.

*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>
  <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" >
  <soa freebytes="87135192" totalbytes="252329472" percent="34" />
  <loa freebytes="8301496" totalbytes="16105984" percent="51" />
</tenured>
</gc>
<tenured freebytes="91937984" totalbytes="268435456" percent="34" >
  <soa freebytes="87135192" totalbytes="252329472" percent="34" />
  <loa freebytes="4802792" totalbytes="16105984" percent="29" />
</tenured>
<time totalms="263.195" />
</af>

```

---

Example 3-5 shows Solaris HotSpot JVM `verbosegc` trace output (young and old).

*Example 3-5 Solaris 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. It is available for free download at IBM alphaWorks® at:

<http://www.alphaworks.ibm.com/tech/pmat>

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 more than one JVM to run 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 58. If your objective is to support large business objects, read 3.4.2, “Tuning for large objects” on page 36.

For most production applications, the IBM JVM Java heap size defaults are too small and should be increased. In general, the HotSpot JVM default heap and nursery size are also too small and should be increased (we 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 IBM JVM on AIX. The essentials can be applied to other systems. Set the initial heap size (`-Xms` option) to something reasonable (for example, 256 MB), and the maximum heap size (`-Xmx`) 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. The output from verbosegc should then be used 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 typically be between a fourth to a half of the total heap size, and it should not be capped at 64 MB. New area sizes are set by JVM options:

- ▶ `-Xmn<size>`
- ▶ `-Xmns<initialSize>`
- ▶ `-Xmnx<maxSize>`

A similar process can be used 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 at least one-fourth of the heap size. The nursery size is set using the `MaxNewSize` and `NewSize` parameters (that is, `XX:MaxNewSize=128m` and `XX:NewSize=128m`).

After the heap sizes are set, `verbosegc` traces should then be used 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 may 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 may not always solve this type of problem, as 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 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:

1. Collect a `verbosegc` trace for each running JVM.
2. 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`).
3. Based on the `verbosegc` trace output, set the maximum heap size appropriately. Care must be taken not to set this value too low, or out-of-memory errors will occur. The maximum heap size must be large enough to allow for peak throughput. Using the above example, 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 (for example, 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.
4. Be careful to not set the heap sizes too low, or garbage collections will occur frequently, which might reduce throughput. Again, a `verbosegc` trace will 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 will, of course, be configuration dependent.



## Reduce or increase heap size if out-of-memory 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 56 for further information about `verbosegc` output. Many garbage collections that produce very little free heap space will generally occur preceding this exception. If this is the problem then one should 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 finalizer count from the `verbosegc` (only the IBM JVM will give this information). If these appear high then a subtle effect may 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. In addition, examine your application to determine whether the finalizers can be avoided or minimized.

Note that out-of-memory errors can also occur for issues unrelated to JVM heap usage, such as running out of certain system resources. If you see an out-of-memory error that is not immediately preceded by a very recent garbage collection, it is highly likely that the cause is not related to JVM heap usage. Examples of this include insufficient file handles or thread stack sizes that are too small. In addition, native (non-JVM) memory is a finite resource. Out-of-memory errors also occur when native memory is exhausted. In some cases, you can tune the configuration to avoid running out of native heap. Try reducing the stack size for threads (the `-Xss` parameter). However, deeply nested methods may force a thread stack overflow in case of insufficient stack size.

For middleware products, if you are using an in-process version of the 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 (for example, DB2's *Net* drivers, Oracle's *Thin* drivers), MQSeries® can be switched from bindings mode to client mode, and so on. Refer to the documentation for the products in question for more details.

## Set AIX threading parameters

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

### *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
```

---

More information about AIX-specific Java tuning can be found at:

<http://www.ibm.com/developerworks/eserver/articles/JavaPart1.html>  
<http://www.ibm.com/developerworks/eserver/articles/JavaPart2.html>

### 3.4.14 Power management tuning

Power management is becoming common in processor technology. Both Intel and Power core processors now have this capability. This capability delivers obvious benefits, but it can also decrease system performance when a system is under high load, so consider whether to enable power management. Using Power6 hardware as an example, ensure that power saver mode is not enabled, unless desired. One way to modify or check this setting on AIX is through the Power Management™ window on the HMC.

### 3.4.15 Tuning for WebSphere InterChange Server migrated applications

Note that the tuning below is unique to applications migrated to WebSphere Process Server using the WebSphere InterChange Server migration wizard in WebSphere Integration Developer. In addition to the tuning specified below, follow the other WebSphere Process Server tuning recommendations detailed in this publication.

- ▶ For JMS-based messaging used to communicate with WBI adapters or custom adapters, make use of non-persistent queues when possible.
- ▶ For JMS-based messaging used to communicate with WBI adapters or custom adapters, make use of WebSphere MQ-based queues if available. By default, the adapters use the MQ APIs to connect to the SIB-based destinations via MQ Link. MQ Link is a protocol translation layer that converts messages to and from MQ-based clients. By switching to WebSphere MQ-based queues, MQLink translation costs will be eliminated and therefore performance will be improved.

### 3.4.16 WebSphere Business Services Fabric tuning

During installation, if the SQLCODE: -964, SQLSTATE: 57011 message is observed, check whether the prerequisite database configuration step has been performed, as described below.

The following are recommended database configuration parameters when using DB2 for the WebSphere Process Server common database. These steps must be carried out before installing Fabric. Before applying any of these settings, back up the database. Connect to the database with an authorized DB user ID and execute the following commands to update database configuration.

```
db2 update db cfg for using logfilsiz 2000
db2 update db cfg for using logprimary 60
db2 update db cfg for using logsecond 100
db2stop force
db2start
```

### 3.4.17 IBM i tuning

This section provides tuning and configuration recommendations that are specific to these products on IBM i. Note that unless indicated otherwise, the tuning options detailed in previous sections also apply to IBM i. The tuning categories are:

- ▶ O/S specific
- ▶ Java Virtual Machine
- ▶ DB2

## O/S-specific tuning

IBM i 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 that 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.

We recommend running 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 ensures 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 that an adequate amount of memory is always available specifically for WebSphere. This is described in the IBM Redbooks publication *WebSphere Application Server for i5/OS Handbook: Version 6.1*, SG24-7221.

## Java Virtual Machine tuning

IBM i supports two different JVMs:

- ▶ The IBM Technology for Java
- ▶ The Classic JVM (See Table 3-6 on page 62 for release-dependent JVM availability.)

The default JVM for the IBM i 6.1 operating system is IBM Technology for Java. It performs better than the Classic JVM. 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 IBM Technology for Java 32-bit JVM for WebSphere Process Server:

```
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 2.5–3.25 GB for most applications. If the application requires a larger heap and you are using IBM i 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, we recommend increasing the initial heap setting. A useful starting point is an initial and max heap size of 1 GB and \*NOMAX, respectively. See 3.4.13, “Advanced Java heap tuning” on page 56, for details about setting 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 "-Djava.compiler=jitc" property. Be sure to always set this property when using the Classic JVM.

Further JVM tuning, JVM performance tools, and recommendations are described in the *System i Performance Capabilities* manual and the *IBM Technology for Java 5.0 Diagnostics Guide* referenced in “Related publications” on page 71.

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

*Table 3-6 Supported and default JVMs by OS level and LPP*

OS level	JVM	LPP	Default JVM	Java_Home
V5R3	Classic-64bit		Classic	/QIBM/ProdData/Java400/jdk*
V5R4	Classic-64bit	5722JV1	Classic	/QIBM/ProdData/Java400/jdk*
V5R4	J9-32bit	5722JV1		/QOpenSys/QIBM/ProdData/JavaVM/jdk*/32bit
V6R1	Classic	5761JV1	J9-32bit	/QIBM/ProdData/Java400/jdk*
V6R1	J9-32bit	5761JV1		/QOpenSys/QIBM/ProdData/JavaVM/jdk*/32bit
V6R1	J9-64bit	5761JV1		/QOpenSys/QIBM/ProdData/JavaVM/jdk*/64bit

\* For the Classic JVM, also set the Java JDK property "java.version=1.4" (5.0 or 6.0).

## DB2 tuning

On IBM i, there are two DB2 drivers that can be used:

- ▶ Native JDBC
- ▶ Toolbox

Native JDBC jobs run in the QSYSWRK subsystem with the job name QSQSRVR. Toolbox jobs run in the QUSRWRK subsystem with the job name QZDASOINIT. We recommend using 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. This may need to be adjusted. You can determine the number of prestarted jobs by running:

```
QSYS/DSFACTPJ SBS(QSYSWRK) PGM(QSQSRVR) {for the native driver}
QSYS/DSFACTPJ SBS(QUSRWRK) PGM(QZDASOINIT) {for the toolbox driver}
```

Once you have determined a suitable number of active database jobs you can use the IBM i Change Prestart Job feature to start these jobs at system startup time, as opposed to runtime (see Example 3-7).

*Example 3-7 Change prestart job feature examples*

---

```
CHGPJE SBSD(QSYSWRK) PGM(QSQSRVR) INLJOBS(20) {set the initial number of jobs to 20}
CHGPJE SBSD(QUSRWRK) PGM(QZDASOINIT) INLJOBS(10) {set the initial number of jobs to 10}
```

---

In order 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.



## Initial configuration settings

In this chapter we recommend 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 30, 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 provides 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 29. 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 30.

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 4-1 Two-tier configuration using JMS file store parameter settings*

Parameter names	BPC DB settings
APP_CTL_HEAP_SZ	144
APPGROUP_MEM_SZ	13001
CATALOGCACHE_SZ	521
CHNGPGS_THRESH	55
DBHEAP	600
LOCKLIST	500
LOCKTIMEOUT	30
LOGBUFSZ	245
LOGFILSIZ	1024
LOGPRIMARY	11

Parameter names	BPC DB settings
LOGSECOND	10
MAXAPPLS	90
MAXLOCKS	57
MINCOMMIT	1
NUM_IOCLEANERS	6
NUM_IOSERVERS	10
PCKCACHESZ	915
SOFTMAX	440
SORTHEAP	228
STMTHEAP	2048
DFT_DEGREE	1
DFT_PREFETCH_SZ	32
UTIL_HEAP_SZ	11663
IMBDEFAULTBP	65536

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 require 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 four 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.
- ▶ The 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 service and remote DB2 server

A three-tier 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 was used for communications. The business process has two modes of operation:

- ▶ A microflow that processed 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)

Three systems were used in this configuration:

- ▶ The request driver
- ▶ The WebSphere Process Server server
- ▶ The DB2 database server

The WebSphere Process Server server and the DB2 database server required more extensive tuning to maximize throughput. Note that some tuning varied due to the operating system (such as AIX and Windows) and the number of processor cores. We present these variations in tabular format after the common tuning:

- ▶ Common WebSphere Process Server server profile creation
  - Production template
  - Security disabled
  - Not run as a service
  - No default or sample applications installed
  - Common database defined as local DB2 type 4
  - Business Process support 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
  - PMI disabled
  - HTTP maxPersistentRequests to -1
  - Java heap minimum and maximum set to 1024 M (See Table 4-2 for any changes.)
  - GC policy was set to -Xgcpolicy:gencon (See Table 4-2 for nursery setting -Xmn.)
  - Remote DB2 databases (connection type 4) used for BPE, SIB System, and SIB BPC databases

Table 4-2 Turning variations

Tuning variations	Microflow: number of processor cores				Macroflow: number of processor cores			
	1	4	8	16	1	4	8	16
Java heap megabytes	1280	1280	1280	1280	1408	1408	1408	1540
Java nursery megabytes-Xmn	768	768	768	768	704	704	704	704
Web container thread pool max	100	200	200	250	100	100	100	100
Default thread pool max	100	200	200	200	200	200	200	200
BPE database connection pool max	150	150	150	150	350	350	350	650
BPC ME database connection pool max	75	75	75	75	75	75	75	75
SYSTEM ME database connection pool max	30	30	30	30	80	80	80	160
Common database connection pool max	80	80	80	80	200	200	200	200
J2C connection factories → BPECF → Connection pools → max connections	40	40	40	40	100	100	100	100



Tuning variations	Microflow: number of processor cores				Macroflow: number of processor cores			
	1	4	8	16	1	4	8	16
J2C Connection factories → BPECFC → Connection pools → max connections	40	40	40	40	100	100	100	100
J2C Connection factories → HTMCF → Connection pools → max connections	20	20	20	20	100	100	100	100
J2C activation specifications → eis/BPEInternalActivationSpec → Custom properties → maxConcurrency	60	60	60	60	160	160	160	160
J2C activation specifications → <i>name</i> → Custom properties → maxConcurrency	40	40	40	80	160	160	160	160
BPEInternalActivationSpec batch size								10
Java custom property com.ibm.websphere.webservices.http.maxConnection	200	200	200	400	200	200	200	200
Application servers → server1 → Business Flow Manager → allowPerformanceOptimization					Yes	Yes	Yes	Yes
Application servers → server1 → Business Flow Manager → interTransactionCache.size					400	400	400	400
Application servers → server1 → Business Flow Manager → workManagerNavigation.messagePoolSize					400	400	400	400
Resources → Asynchronous Beans → Work Managers → BPENavigationWorkManager → min threads, max threads, request queue size					20, 200, 400	20, 200, 400	20, 200, 400	20, 300, 400
Application servers → server1 → Business Process Choreographer → Business Flow Manager → Custom Properties → DataCompressionOptimization					false	false	false	false

For details of enableInProcessConnections, go to:

[http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp?topic=/com.ibm.websphere.base.doc/info/aes/ae/rrun\\_inbound.html](http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp?topic=/com.ibm.websphere.base.doc/info/aes/ae/rrun_inbound.html)

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

- ▶ The SCA.SYSTEM.*cellname*.BUS database:  

```
db2 update db cfg for sysdb using logbufsz 512 logfilsiz 8000 logprimary 20  
logsecond 20 auto_runstats off  
db2 alter bufferpool ibmdefaultbp size 30000
```
- ▶ The BPE database was created and tuned as follows:  

```
db2 CREATE DATABASE bpedb ON /raid USING CODESET UTF-8 TERRITORY en-us  
db2 update db cfg for bpedb using logbufsz 512 logfilsiz 10000 logprimary 20  
logsecond 10 auto_runstats off
```
- ▶ Using the WebSphere Process Server generated script: **db2 -tf createTablespace.sql**
- ▶ Using the WebSphere Process Server generated script: **db2 -tf createSchema.sql**  

```
db2 alter bufferpool ibmdefaultbp size 132000  
db2 alter bufferpool bpebp8k size 132000
```

## 4.2 WebSphere ESB 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 29. 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 30 for a discussion of tuning methodology.

### 4.2.1 WebSphere ESB common settings

These 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.
- ▶ Gencon garbage collection policy is enabled, setting the nursery heap size to 1024 MB.

### 4.2.2 WebSphere ESB settings for Web services

The WebSphere ESB settings for Web services are:

- ▶ PMI monitoring is disabled.
- ▶ WebContainer thread pool sizes set to max 50 and min 10.
- ▶ WebContainer thread pool inactivity timeouts for thread pools set to 3500.

### 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

These settings are only relevant for JMS persistent configurations as they make use of the database to persist messages:

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

### 4.3 WebSphere Business Monitor database settings

We recommend using the following DB2 commands to tune the Monitor database:

```
db2 update db cfg for Monitor using logprimary 128 logfilsiz 8192 logbufsz 128
db2 update db cfg for Monitor using auto_maint off
```



# Related publications

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

## IBM Redbooks publications

For information about ordering these publications, see “How to get Redbooks” on page 72. Note that some of the publications referenced here may be available in softcopy only.

- ▶ *Production Topologies for WebSphere Process Server and WebSphere ESB V6*, SG24-7413
- ▶ *WebSphere Application Server for i5/OS Handbook: Version 6.1*, SG24-7221
- ▶ *IBM WebSphere InterChange Server Migration to WebSphere Process Server*, SG24-7415
- ▶ *WebSphere Application Server for i5/OS Handbook: Version 6.1*, SG24-7221
- ▶ *Configuring IBM WebSphere Process Server V6.1 with an Oracle Database*, REDP-4432
- ▶ *IBM WebSphere Business Process Management V6.1 Performance Tuning*, REDP-4431

## Online resources

These Web sites are also relevant as further information sources:

- ▶ WebSphere Application Server Performance Web site  
<http://www.ibm.com/software/webservers/appserv/was/performance.html>
- ▶ WebSphere Application Server 6.1 information center (including Tuning Guide) Web page  
[http://www.ibm.com/software/webservers/appserv/was/library/?S\\_CMP=rnav](http://www.ibm.com/software/webservers/appserv/was/library/?S_CMP=rnav)
- ▶ WebSphere BPM Version 6.2.0 information center (product documentation for WebSphere Process Server, WebSphere ESB, WebSphere Integration Developer, Monitor, and Fabric)  
[http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r2mx/index.jsp?topic=/com.ibm.websphere.wps.620.doc/welcome\\_wps.html](http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r2mx/index.jsp?topic=/com.ibm.websphere.wps.620.doc/welcome_wps.html)
- ▶ Information Center topic: Setting up a Data Store in the Messaging Engine  
[http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.pmc.nd.doc/tasks/tjm0030\\_.html](http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.pmc.nd.doc/tasks/tjm0030_.html)
- ▶ IBM DeveloperWorks paper: *Best practices for DB2 for Linux, UNIX, and Windows* paper  
[http://www.ibm.com/developerworks/data/bestpractices/?S\\_TACT=105AGX11&S\\_CMP=FP](http://www.ibm.com/developerworks/data/bestpractices/?S_TACT=105AGX11&S_CMP=FP)
- ▶ IBM DB2 Database for Linux, UNIX, and Windows Information Center  
<http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/index.jsp>
- ▶ *WebSphere Process Server V6.1 - Business Process Choreographer: Performance Tuning Automatic Business Processes for Production Scenarios with DB2*  
<http://www.ibm.com/support/docview.wss?uid=swg27012639&aid=1>

- ▶ PA71: WebSphere Process Server - Query Table Builder Web document  
<http://www.ibm.com/support/docview.wss?uid=swg24021440>
- ▶ Query tables in Business Process Choreographer Web document  
[http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r2mx/index.jsp?topic=/com.ibm.websphere.bpc.620.doc/doc/bpc/c6bpe1\\_querytables.html](http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r2mx/index.jsp?topic=/com.ibm.websphere.bpc.620.doc/doc/bpc/c6bpe1_querytables.html)
- ▶ *Extending a J2CA adapter for use with WebSphere Process Server and WebSphere Enterprise Service Bus*  
<http://www.ibm.com/developerworks/library/ws-soa-j2caadapter/index.html?ca=drs->
- ▶ WebSphere Process Server Support Web page  
<http://www.ibm.com/software/integration/wps/support/>
- ▶ WebSphere Enterprise Service Bus Support Web page  
<http://www.ibm.com/software/integration/wsesb/support/>
- ▶ IBM Java 5.0 Diagnostic Guide  
<http://publib.boulder.ibm.com/infocenter/javasdk/v5r0/index.jsp>
- ▶ Oracle Diagnosing Performance Problems Web information  
[http://www.oracle.com/technology/oramag/oracle/04-jan/o14tech\\_perf.html](http://www.oracle.com/technology/oramag/oracle/04-jan/o14tech_perf.html)
- ▶ Oracle 10g Release 2 documentation (includes a Performance Tuning Guide)  
<http://www.oracle.com/pls/db102/homepage>
- ▶ Oracle Database 11g Release 1 (11.1) Documentation  
<http://www.oracle.com/technology/documentation/database11gR1.html>
- ▶ IBM DB2 for i home page  
<http://www.ibm.com/systems/i/software/db2/>
- ▶ IBM PartnerWorld white papers and technical documentation for WebSphere Business Process Management on IBM i  
<http://www.ibm.com/jct09002c/partnerworld/wps/servlet/ContentHandler/W980921J36739H59>
- ▶ Performance Management on IBM i Resource Library  
<http://www.ibm.com/systems/i/advantages/perfmgmt/resource.html>

## How to get Redbooks

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

[ibm.com/redbooks](http://ibm.com/redbooks)

## Help from IBM

IBM Support and downloads

[ibm.com/support](https://ibm.com/support)

IBM Global Services

[ibm.com/services](https://ibm.com/services)









# WebSphere Business Process Management 6.2.0 Performance Tuning



**Learn valuable tips for tuning**

**Get the latest best practices**

**Try the example settings**

This IBM® Redpaper publication was produced by the IBM WebSphere® BPM performance teams. It provides performance tuning tips and best practices for the following products:

- ▶ WebSphere Process Server 6.2.0
- ▶ WebSphere Enterprise Service Bus 6.2.0
- ▶ WebSphere Integration Developer 6.2.0
- ▶ WebSphere Business Monitor 6.2.0
- ▶ WebSphere Business Services Fabric 6.2.0

These products represent an integrated development and runtime environment based on a key set of service-oriented architecture (SOA) and Business Process Management (BPM) technologies: Service Component Architecture (SCA), Service Data Object (SDO), and Business Process Execution Language for Web Services (BPEL). These technologies in turn build on the core capabilities of the WebSphere Application Server 6.1.

For those who are either considering or are in the very early stages of implementing a solution incorporating these products, this publication provides best practices for application development and deployment, and setup, tuning and configuration information. It provides a useful introduction to many of the issues influencing each product's performance, and could act as a guide for making rational first choices in terms of configuration and performance settings.

Finally, these products build on the capabilities of WebSphere Application Server, so consult tuning, configuration, and best practices information for WebSphere Application Server and corresponding platform JVMs (documented in the Related Publications chapter).

## INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

### BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

**For more information:**  
[ibm.com/redbooks](http://ibm.com/redbooks)