WebSphere. software

Draft Document for Review April 9, 2010 12:16 pm

REDP-4664-00

WebSphere Business Process Management (BPM) V7 Performance

Learn valuable tips for tuning

Get the latest best practices

Try the example settings

WebSphere Business Process Management Performance Team

Redpaper

ibm.com/redbooks

International Technical Support Organization

WebSphere Business Process Management (BPM) V7 Performance Tuning

April 2010

REDP-4664-00

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

First Edition (April 2010)

This edition applies to VWebSphere Process Server 7.0.0.1, WebSphere Enterprise Service Bus 7.0.0.1, WebSphere Integration Developer 7.0.0.1, WebSphere Business Monitor 7.0.0.0, WebSphere Business Modeler 7.0.0.1.

This document created or updated on April 9, 2010.

Contents

Notices

Trademarks

Preface

This IBM® Redpaper™ publication was produced by the WebSphere® Business Process Management performance teams. It provides performance tuning tips and best practices for WebSphere Process Server 7.0.0.1, WebSphere Enterprise Service Bus 7.0.0.1, WebSphere Integration Developer 7.0.0.1, WebSphere Business Monitor 7.0.0.0, and WebSphere Business Modeler 7.0.0.1. 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).

We envision this paper to be read by a wide variety of groups, both within IBM (development, services, technical sales, etc.) and by customers. For those who are either considering or are in the very early stages of implementing a solution incorporating these products, this document should prove a useful reference, both in terms of best practices during application development and deployment, and as a reference for setup, tuning and configuration information. This paper provides a useful introduction to many of the issues influencing each product's performance, and can serve as a guide for making rational first choices in terms of configuration and performance settings. Similarly, those who have already implemented a solution utilizing these products might effectively use the information presented here to gain insight as to how their overall integrated solution performance may be improved.

All of these products build on the core capabilities of the WebSphere Application Server infrastructure, so BPM solutions also benefit from tuning, configuration, and best practices information for WebSphere Application Server and the corresponding platform Java™ Virtual Machines (JVMs). Pointers to this information can be found in the Related publications. The reader is encouraged to use this paper in conjunction with these references.

Products covered by this publication

A short description of each product covered in this paper follows:

- WebSphere Process Server allows the deployment of standards-based business integration applications in a service-oriented architecture (SOA), which takes everyday business applications and breaks them down into individual business functions and processes, rendering them as services. Based on the robust J2EE 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 (WebSphere ESB) 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 (SOA)-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 Modeler is IBM's premier business process modeling and analysis tool for business users. It offers process modeling, simulation, and analysis capabilities to help business users understand, document, and deploy business processes for continuous improvement.

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 who wrote this paper

This paper is authored by the IBM WebSphere BPM performance team, with members in Austin Texas, Böblingen Germany, and Hursley England.

- Rajiv Arora
- Mike Collins
- Weiming Gu
- Kean Kuiper
- Ben Hoflich
- Chris Richardson
- Randall Theobald
- Steve Garde
- Phani Achanta
- Paul Harris
- Sam Massey

- Jonas Grundler

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author - all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

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

  ibm.com/redbooks

- Send your comments in an e-mail to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Architecture best practices

This section provides guidance on how to architect a high-performing and scalable WebSphere BPM solution. The purpose of this chapter is to highlight the best practices associated specifically with the technologies and features delivered in the WebSphere BPM products covered in this paper. However, these products are built on top of existing technologies like WebSphere Application Server and DB2. Each of these technologies has associated best practices that apply. It is not our intent to enumerate these here. Instead the reader is referred to “Related publications” on page 67 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 WebSphere BPM solutions. Development best practices and performance tuning and configuration are covered in subsequent chapters. The reader is strongly encouraged to read these chapters, since the authors have found this information to be very beneficial for numerous customers over the years. ***However, if you read nothing else in this document, please 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 (e.g. RAID adapter with multiple physical disks) is required on the tier(s) that host the message and data stores to achieve acceptable performance. This point cannot be overstated; the authors have seen many cases where the overall performance of a solution is 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 3.3.2, "Java tuning parameters" on page 31.
- Where possible, utilize non-interruptible processes (microflows) instead of long running processes (macroflows). Macroflows are required for many processes (e.g, if human tasks are employed, or state needs to be persisted). However, there is significant performance overhead associated with macroflows. Further, if macroflows are needed for some portion of the solution, separate the solution into both microflows and macroflows to maximize utilization of microflows. For details, see "Choose non-interruptible processes whenever possible" on page 3.
- Use DB2 instead of the default Derby DBMS. DB2 is a high-performing, industrial strength database designed to handle high levels of throughput and concurrency, scale well, and deliver excellent response time.
- Tune your database for optimal performance. Proper tuning, and deployment, choices for databases can greatly increase overall system throughput. For details, see 3.4.10, "Database: general tuning" on page 46.
- Disable tracing. Tracing is clearly important when debugging, but the overhead of tracing severely impacts performance. More information is available in 3.4.1, "Tracing and monitoring considerations" on page 34.
- Configure thread and connection pools to enable 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 36.
- 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 list 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 Work-Manager-based navigation for business processes " on page 42.
- Avoid unnecessary usages 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 19.

- Avoid too granular transaction boundaries in SCA and BPEL. Every transaction commit results in expensive database and/or messaging operations. Design your transactions with care, as described in “Transactionality considerations” on page 17.

1.2 Modeling

This section describes best practices for modeling.

1.2.1 Choose non-interruptible processes whenever possible

Use interruptible processes, a.k.a. macroflows or long running processes, only when required (e.g. long running service invocations and human tasks). Non-interruptible processes, also referred to 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. A non-interruptible process is much more efficient than an interruptible process since it does not have to utilize state or persistence in the backing database system.

A process is interruptible if the checkbox “Process is long-running” is set in the 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 list and process list queries

Query tables were 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 a query tables that reflect the information which is displayed on task lists and process lists presented to users.

For further information, please 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 in the WebSphere Process Server 7.0 Information Center:

  http://publib.boulder.ibm.com/infocenter/dmndhelp/v7r0mx/index.jsp?topic=/com.ibm.websphere.bpc.doc/doc/bpc/c6bpel_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 POJOs (Plain Old Java Objects) 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. Further, emitting CBEs to a database is not recommended; 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 holds for most customers:

- Customers are concerned about the state of their business and their processes. Therefore events that signify changes in state are important. For long-running and human task activities, this is fairly natural: use events to track when long-running activities complete, when human tasks change state, etc.
- 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, end) are usually sufficient for a microflow.

1.2.5 Choose efficient metadata management

This section describes best practices for metadata usage.

Follow Java language specification for complex data type names

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

Avoid use of anonymous derived types in XSDs

Some XSD features (restrictions on the primitive string type, for example) result in modifications to the type that require a new sub-type to be generated. If these types are not explicitly declared, then a new sub-type (a derived type) is generated at runtime. Performance is generally better if this can be avoided. So, avoid adding restrictions to elements of primitive type where possible. If a restriction is unavoidable, consider creating a new, concrete SimpleType that extends the primitive type to include the restriction. Then XSD elements may utilize that type without degraded performance.

Avoid referencing elements from one XSD in another XSD

For example, 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 is shown in Example 1-3.

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 is shown in Example 1-4.

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 Considerations when choosing between business state machines and business processes

Business state machines (BSMs) 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. However, a 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 BSMs

Where possible, minimize external events to drive state transitions in BSMs. 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 BSM. For example, if a state machine transitions through states A → B → B → B → C, (four transitions), it is twice as time consuming as making transitions through states A → B → 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 in picking a hardware configuration:

- **Cores**: Ensure that WebSphere Process Server and WebSphere ESB are installed on a modern server system with multiple cores. Both products 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 as well as an ample amount of physical memory. Ensure that the chosen system has server-class memory controllers and as large as possible L2 and L3 caches (optimally, use a system with at least a 4 MB L3 cache). Make sure there is enough physical memory for all the applications (JVMs) combined that are expected to run concurrently on the system. 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 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, and Monitor 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's DB2) for any performance measurement or production installation. Examples of databases that can be moved to DB2 include the BPE database, relationship databases, and the WebSphere Platform Messaging (WPM) messaging engine data stores.

The conversion requires the administrator to 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 the IBM Redbook publication *WebSphere Business Process Management V7 Production Topologies*, SG24-7854 to our readers, 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 remote messaging and remote support deployment pattern

Use the remote messaging and remote support deployment environment pattern for maximum flexibility in scaling. See *Deployment environment patterns* at:
http://publib.boulder.ibm.com/infocenter/dmndhelp/v7r0mx/index.jsp?topic=/com.ibm.websphere.wps.doc/doc/cpln_topologypat.html

This topology (formerly known as the Golden Topology) prescribes the use of separate clusters for applications, messaging engines, and support applications like the CEI (Common Event Infrastructure) server, and the Business Rules Manager. 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 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 the reader is familiar with these kinds of system tradeoffs, as they occur in most server middleware.

Single server vs. clustered topology considerations

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

- Scalability and load balancing in order to improve overall performance and throughput
- High availability by using failover to another cluster member to prevent loss of service due to hardware or software failures.

Although not mutually exclusive, there are considerations applicable to each. In this paper, the focus is on the performance (throughput) related aspects of clustering, and not on the high availability aspects. Most single server workloads that are driving resources to saturation would benefit to some degree by moving to a clustered topology.

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 it. 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 process server components that invoke this application should be tuned so that the simultaneous request 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 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 such 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 business process, and its resources, until the 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 as well as practical guidelines for the current releases of these products.

The single most important factor affecting large object processing is the JVM. Note that there was a significant change in JVM technology starting with Java 5, which was first used by the

WebSphere BPM products in V6.1.0. WebSphere BPM v7 utilizes the Java 6 JVM, which is the follow-on release to Java 5. Due to the changes in the underlying JVM technology, this section has been rewritten and the recommendations and best practices differ from those in WebSphere BPM 6.0.2 and earlier, which used the 1.4.2 JVM.

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:

1. 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 “Heap limitations: increase the Java heap to its maximum” on page 34, 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.

2. Size of in-memory business objects

   Business objects, when represented as Java objects, are much larger in size than when represented in wire format. For example, a business object that consumes 10 MB on an input JMS message queue 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 business object flows through the adapters and WebSphere Process Server or WebSphere ESB. There are a number of factors that affect the in-memory expansion of business objects.

   – The single-byte binary wire representation is generally converted to multi-byte character representations (e.g. Unicode), resulting an expansion factor of 2.

   – The business object 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-bye or 16-byte address boundaries.

   – As the business object 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 business object copies in order for the transaction to complete successfully.

3. 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 has to 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 “64-bit considerations ” on page 10 for details.

Batched inputs: send large objects as multiple small objects

If a large object needs to 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, 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: only a small portion of an input message is used

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

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

The Claim-Check pattern requires custom code and snippets in the solution. A less developer-intensive variant would be to make use of custom data bindings to generate the control business object. This approach suffers from the disadvantage of being limited to certain export and import bindings, and the full payload still must be allocated in the JVM.

1.5 64-bit considerations

Full 64-bit support has been available since WebSphere Process Server 6.1.0. 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 4GB 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.

BPM V7 brought further improvement to its 64-bit implementation. The memory footprint of a 64-bit runtime server is now about the same as the 32-bit version. What this means is that there is no longer a memory footprint penalty for utilizing 64-bit if the heap size is lower than 27 GB. This was not the case for BPM v6.1 and v6.2.

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

- 64-bit mode is an excellent choice for applications whose liveset approaches or exceeds the 32-bit limits. Such applications either experience OutOfMemoryExceptions or suffer excessive time in GC. We consider anything > 10% of time in GC as excessive. These applications will exhibit much better performance when allowed to run with the larger heaps they need. However, there must always be sufficient physical memory on the system to back the Java heap size.
- 64-bit mode is also 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 tradeoff the more space available in 64-bit heaps for less execution time would yield much better performance.
- Moving to 64-bit still causes some degradation in throughput. If a 32-bit application fits well within a 1.5-2.5GB heap, and the application is not expected to grow significantly, 32-bit WebSphere Process Server BPM servers can still be a better choice than 64-bit.

1.6 WebSphere Business Monitor

This section describes best practices for performance with 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.

By default, events are delivered directly from CEI to the monitor database, bypassing an intermediate queue. We recommend using this default delivery style for better performance, as it avoids an additional persistence step in the flow. For additional background see *Bypassing the JMS Queue* in the WebSphere Business Monitor Information Center at:

`http://publib.boulder.ibm.com/infocenter/dmndhelp/v7r0mx/topic/com.ibm.btools.help.monitor.inst.doc/inst/cfg_qb.html`

1.6.2 Dashboard

The platform requirements of the Business Space, Dashboard, and Alphablox stack are relatively modest compared to those of Monitor server and the database server. The most important consideration for good Dashboard performance is to size and configure the database server correctly. Be sure it has enough CPU capacity for anticipated data mining queries, enough RAM for bufferpools etc., 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 database administrator skills. We strongly advise that the database tuning advice and recommendations beginning in 3.4.10, “Database: general tuning” on page 46 be read and followed.

Development best practices

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

2.1 Service Component Architecture (SCA) considerations

This section discusses the SCA considerations for performance tuning.

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,* i.e. 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, i.e. 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 considerations for performance tuning specific to business processes.

2.2.1 Modeling best practices for activities in a business process

Use the following guidelines when modeling a business process:

- Use the “Audit Logging property for Business Processes only” setting 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 the process and for each individual BPEL activity. Enabling this flag causes details 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 “Avoid two-way synchronous invocation of an asynchronous target” on page 17 for more details.
- Human tasks can be specified in business processes (e.g. process administrators), invoke activities, and receive activities. Specify human 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, etc.) 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 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 business objects

Follow these guidelines when defining BPEL variables and business objects:

- Use as few variables as possible and minimize the size and the number of business objects used. In long-running processes, each commit saves modified variables to the database (to save context), and multiple variables or large business objects make this very costly. Smaller business objects 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 business objects 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).
- 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. This improves the response time of "task complete" API calls.

2.4 Business process and human tasks client considerations

The following are general considerations for 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 to continue processing.
- Chose the protocol which best suits your needs:
  - In a J2EE environment, use the HTM and BFM EJB APIs. If the client application is running on a WebSphere Process Server, use the local EJB interface.
  - In a Web 2.0 application, use the REST API.
  - In an application that runs remote to the process container, the Web services API is an option.

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

- Use the "completeAndClaimSuccessor()" API if possible. This provides optimal response time.
- Applications that assign the next available task to the user can use the claim(String queryTableName, …) method on the Human Task Manger EJB interface. It implements a performance optimized mechanism to handle claim collisions.
- Don't 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 invoke long-running sub-processes between two steps of a page-flow, because long-running 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.
- Do not loop over the tasks displayed in the task or process list *and* execute an additional remote call for each object. This will prevent 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 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.
- Avoid synchronous invocation of a two-way asynchronous target. The caller transaction's failure recovery can be problematic.

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 following "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 path assumes that the first component in such a chain either starts or participates in a global transaction.

2.5.2 Avoid two-way synchronous invocation of an asynchronous target

If the target component has to be invoked asynchronously and its interface is of two-way request/response style, the target cannot be safely invoked through synchronous SCA calls. After the caller sends the request to the target, it then waits for response from the target. Upon receiving the request, the asynchronous target starts a new transaction, and upon completion of the request processing returns the response asynchronously to the caller through the response queue. If a system failure occurs after the caller successfully sends the request but before receiving the response, the caller transaction is rolled back and then retried. As a result, the target will be invoked a second time.

2.5.3 Exploit transactional attributes for BPEL activities in long-running processes

While SCA qualifiers control component level transactional behavior, there are additional transactional considerations in long-running business processes which 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 *Transactional behavior of business processes* in the WebSphere Process Server Information Center for a detailed description of these settings:

`http://publib.boulder.ibm.com/infocenter/dmndhelp/v7r0mx/index.jsp?topic=/com.ibm.websphere.bpc.doc/doc/bpc/cprocess_transaction.html`

There are four choices for transactional behavior settings:

- 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. Please refer to the Information Center article 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 longer response time to the user. This is a classic response time vs. throughput tradeoff.

Note that starting with the 6.2.0 release, Receive and Pick activities in BPEL flow are 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 that behavior will perforrn 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, it is recommended to set the Preferred Interaction Style setting 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 of the WebSphere Integration Developer, 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/v7r0mx/index.jsp?topic=/com.ibm.websphere.bpc.doc/doc/bpc/cprocess_transaction.html

Some additional considerations are listed below:

- When setting an interface's Preferred interaction style to asyn , 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 at `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.

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.

Some qualities of services such as event sequencing and store-and-forward can only be associated with asynchronous SCA calls. Consider the performance impact of asynchronous invocations when setting these qualities of service.

2.6.3 Avoid asynchronous invocation of synchronous services in a FanOut / FanIn block

Do not select asynchronous (deferred response interaction) service invocations for services with synchronous bindings (e.g. Web services) unless there is an overriding need for this, and the non-performance implications for this style of invocation are well understood.

Apart from the performance implications of calling a synchronous service asynchronously there are reliability and transactional aspects to be considered. Make sure you understand these non-performance implications of using asynchronous callouts before considering their use. Generally, asynchronous callouts should only be used for idempotent query type services. If you need to guarantee that the service is only called once do not use asynchronous invocation. It is beyond the scope of this paper to provide complete guidance on the functional applicability of using asynchronous callouts in your mediation flow; more information can be found in the WebSphere Integration Developer help documentation and WebSphere Process Server/WebSphere ESB Information Centers.

Assuming that asynchronous callouts are functionally applicable for you there may be a performance reason for invoking a service in this style but it should be understood that asynchronous processing is inherently more expensive in terms of CPU cycles due to the additional messaging overhead incurred by calling a service this way.

There are additional operational considerations, for example asynchronous invocations use the service integration bus messaging infrastructure which uses a database for persistence. Synchronous invocations will perform well with basic tuning of the JVM heap size and thread pools but for asynchronous invocations SCA artifacts require review and tuning. This will include tuning of the SCA messaging engine (see section 3.3.7, “Messaging engine properties” on page 33), data sources (section 3.3.6, “JDBC data source parameters” on page 32) and the database itself. For the data source, the tunings for JMS bindings in this paper can be used as guidance as the considerations are the same.

If multiple synchronous services with large latencies are being called then asynchronous invocations can reduce the overall response time of the mediation flow at the expense of increasing the internal response time of each individual service call. This assumes that asynchronous callouts have been configured along with parallel waiting in the FanOut section of the flow:

- In the case of iteration of array, configuring the FanOut to "check for asynchronous responses after all/N messages have been fired"
- In case of extra wires/FlowOrder primitive - by default.

If there are a number of services in a fan-out section of a mediation flow then calling these synchronously will result in an overall response time equal to the sum of the individual service response times.

Calling the services asynchronously (with parallel waiting configured) will result in a response time equal to at least the largest individual service response time in WebSphere ESB plus the sum of the time taken by WebSphere ESB to process the remaining service callout responses residing on the messaging engine queue.

For a FanOut/FanIn block the processing time for any primitives before or after the service invocations will need to be added in both cases.

To optimize the overall response time when calling services asynchronously in a FanOut/FanIn section of a mediation flow you should invoke the services in the order of expected latency if known (highest latency first).

There is a trade off between parallelism and additional asynchronous processing to consider. The suitability of asynchronous processing will depend on the size of the messages being processed, the latency of the target services, the number of services being invoked and any response time requirements expressed in service level agreements. Running performance evaluations on mediation flows including fan-outs with high latency services is strongly recommended if asynchronous invocations are being considered.

The default quality of service on service references is Assured Persistent. A substantial reduction in asynchronous processing time can be gained by changing this to Best Effort (non-persistent) which eliminates I/O to the persistence store but the application MUST tolerate the possibility of lost request or response messages. This level of reliability for the service integration bus can discard messages under load and may require tuning.

2.7 Mediation flow considerations

This section discusses mediation flow considerations for performance.

2.7.1 Use mediations that benefit from WebSphere ESB optimizations

Certain types of mediations benefit from internal optimization in WebSphere ESB, and deliver improved performance. This specialized optimization can be regarded as a kind of 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
  - Business Object (BO) Mapper
  - Fan Out
  - Fan In
  - Set Message Type
  - Message Logger
  - Event Emitter (Except for CBE Header only)
  - EndpointLookup utilising 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 fastpath'ed 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 these optimizations, the ordering of primitives can be important. If the mediation flow contains an XSLT primitive (with a root of /body, meaning it is in the category 1 variant) and category 3 primitives then the XSLT primitive should be placed ahead of the other primitives. This means that

```
→ 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 or not. 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 which may not be optimally written. The above guidelines can help in designing for performance but they cannot guarantee speed.

2.7.2 Usage of XSLTs vs. business object maps

In a mediation flow that is eligible for deferred parsing (detailed above), the XSL Transform primitive gives 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 gives better performance than the XSL Transform primitive.

Note that if you are transforming from the root (/), then the Business Object Map will always perform better.

2.7.3 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 and tuning scripts to be applied
- Easier post deployment adjustment

Tooling-created resources support:

- No further need for creating resources using scripts or the Admin Console

- The ability to change the majority of performance tuning options as they now exposed in the Tooling

In our performance tests we use pre-configured resources for the reason that 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 was not followed in our tests was for generic JMS bindings. In these scenarios where resources have already been configured by the third-party JMS provider software (MQ 6.0.2.2 for all instances in this paper), the tooling-created resources were used to locate the externally defined resources.

2.8 Large object best practices

This section discusses best practices for performance when using large objects.

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 business objects

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

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 business object size is large. To avoid unnecessarily creating a business object 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 which expand their capacity based on input (for example, StringBuffer and 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 document in their native form and avoid inflating them into their full business object representation. These mediations are listed above in “Use mediations that benefit from WebSphere ESB optimizations” on page 21. 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 WebSphere Business Integration adapters (WBI adapaters), where possible. Migrated workloads making use of custom WBI adapters or legacy WBI adapters result in interaction with the WebSphere Process 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 an SCA binding will remove the dependency on a legacy adapter as well as have better performance.
- The WebSphere InterChange Server Migration Wizard in WebSphere Integration Developer 7.0 offers a feature to merge the connector and collaboration module together. Enable this option, if possible, as it increases performance by reducing cross-module SCA calls.
- WebSphere InterChange Server Collaborations are migrated into WebSphere Process Server BPEL processes. The resultant BPEL processes can be further customized and made more efficient.
  - Migrated BPEL processes enable support for compensation by default. If the migrated workload does not make use of compensation, this support can be disabled to gain performance. The relevant flag can be found in the WebSphere Integration Developer under ***process name*** → **properties** → **Details** → **Require a compensation sphere context to be passed in.**
  - The generated BPEL flows still make use of the ICS APIs to perform business object and collaboration level tasks. Development effort spent cleaning up the migrated BPEL to use WebSphere Process Server APIs instead of the ICS APIs 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, for example business rules, may be better choices. Investigate the BPEL processes produced by migration to ensure the processes are the best fit for your scenario.
- Disable Message Logger calls in migration-generated mediation flow component (MFC) components. The WebSphere InterChange Server Migration Wizard in WebSphere Integration Developer 7.0 generates an MFC to deal with the mapping details of a connector; it contains the code for handling synchronous and asynchronous calls to maps that transform application specific business objects to generic business objects and visa versa. The generated MFC contains embedded MessageLogger calls that log the message to a database. Disable these calls if not required in your business scenario to

reduce writes to the database and thus improve performance. (Select the MessageLogger instance, choose the Details panel, then uncheck the “Enabled” checkbox.)

- 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 for 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 to only reference the shared libraries that they need.
- If the original WebSphere InterChange Server maps contain many custom map steps, then the development effort spent in rewriting those map steps will result in better performance. The WebSphere InterChange Server Migration Wizard in WebSphere Integration Developer 7.0 generates maps that make use of the ICS APIs, which is a translation layer above WebSphere Process Server technologies. Removing this layer by making direct use of WebSphere Process Server APIs would avoid the cost of translation and hence produce better performance.

2.10 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.

2.10.1 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. Also, for I/O intensive activities (like Import) a faster disk drive or disk subsystem reduces total response time.

2.10.2 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 WebSphere Application Server shared library as described in the technote found at:

http://www-01.ibm.com/support/docview.wss?uid=swg21298478

2.11 Fabric considerations

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

2.11.1 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 don’t list that context property in the associated context specification.

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

2.11.2 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 which 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 should then be placed in one of these categories and the category should be passed as the context instead of the actual values.

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, like 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 paper.

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

The first 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. In addition, representative values for many of the tuning parameters are shown in the next chapter, Initial Configuration settings.

Important: 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, "Related publications" on page 67 contains referencees to related documentation 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. Please note that the art of system performance tuning, which requires training and experience, is not going to be exhaustively described here. Rather, we will highlight some key aspects of tuning that are particularly important.

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

- Physical hardware topology choices
- Operating System parameters tuning
- WebSphere Process Server, WebSphere Application Server, and messaging engine 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 will now examine each in turn:

1. Pick a set of reasonable initial parameter settings.
   a. Use the tuning checklist in “Tuning checklist” on page 29 for a systematic way to set parameters.
   b. For specific initial values, consult Chapter 4, “Initial configuration settings” on page 59 for settings that were used for the various workloads that were used for our internal performance evaluation. These values can be considered for initial values.
2. Monitor the system. We recommend monitoring the system(s) 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 database servers:
     • Monitor processor core utilization, memory utilization, disk utilization, network utilization using relevant OS tools like vmstat, iostat, netstat, or equivalent
   – For each JVM process started on a physical machine, i.e. process server, messaging engine server, etc.
     • Use tools like “ps” or equivalent to get core and memory usage per process
     • Collect verbosegc statistics
   – For each WebSphere Process Server or messaging engine JVM, use Tivoli® Performance Viewer 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 which 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 etc. These can be resolved either by adding more physical resources, or rebalancing the load more evenly across the available resources.
- Excessive utilization of virtual resources. Examples include heap memory, connection pools, thread pools, etc. 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.
  - If security is required use Application security, not Java2 security.
  - Use appropriate hardware configuration for performance measurement, for example, laptops 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 production server in “development mode” or with “development profile”.
  - Do not use the Unit Test Environment (UTE) for performance measurement.
  - Tune external service providers and external interfaces to ensure they are not the system bottleneck.
  - Configure message-driven bean activation specifications
  - Configure for clustering (where applicable)
  - Configure thread pool sizes
  - Configure data sources settings for connection pool size and 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
    - If work-manager based navigation is used, also optimize message pool size and intertransaction cache size
  - Use query tables to optimize query response time
  - Optimize Business Flow Manager resources: database connection (BPEDB), activation specification (BPEInternalActivationSpec), and JMS connection (BPECF and BPECFC);
  - 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 , e.g. turn off audit logging
- Messaging and message bindings
  - Optimize activation specification (JMS, MQJMS, MQ)
  - Optimize queue connection factory (JMS, MQJMS, MQ)
  - Configure connection pool size (JMS, MQJMS, MQ)
  - Configure service integration bus data buffer Sizes
- Database
  - Place database tablespaces and logs on a fast disk subsystem
  - Place logs on a separate device from the 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 and nursery sizes to manage memory efficiently
  - Choose the appropriate garbage collection policy (generally, -Xgcpolicy:gencon)
- 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 the paper.

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.

This section lists tracing parameters used in the products covered in this paper. Some flags or checkboxes 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 using the administrative console.

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

To change the PMI level go to Monitoring and **Tuning** → **Performance Monitoring Infrastructure** → ***server_name*** and select `none`.

In addition, Cross-Component Tracing (XCT) is very useful for problem determination, enabling correlation of SCA component information with log entries. However, XCT should not be used in production or while obtaining performance data. Further, there are two levels of XCT settings: “enable” or “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 the Configuration or Runtime tab. Changes made on the Runtime tab take effect immediately while changes made on the Configuration tab 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 garbage collection (GC) policy

The default garbage collection algorithm on platforms with an IBM JVM is a generational concurrent collector (specified with `-Xgcpolicy:gencon` under the Generic JVM arguments on the Java Virtual Machine administrative console panel). Our internal evaluation shows that this garbage collection policy usually delivers better performance with a tuned nursery size as discussed in the next section.

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 panel in the administrative console.

If Generational Concurrent Garbage Collector is used, the Java heap is divided into a new area (nursery) where new objects are allocated and an old area (tenured space) where longer lived objects reside. The total heap size is the sum of the new area and the tenured space. The new area size can be set independently from the total heap size. Typically the new area size should be set between ¼ and ½ of the total heap size. The relevant parameters are:

- Xmns<size> : initial new area size
- Xmnx<size> : maximum new area size
- Xmn<size> : fixed new area size

3.3.3 MDB ActivationSpec

There are a few shortcuts you can take in the administrative console 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***

The following two custom properties in the MDB ActivationSpec have considerable performance implications. These are discussed further in “Tune MDB ActivationSpec properties” on page 36 .

- maxConcurrency
- maxBatchSize

3.3.4 Thread pool sizes

WebSphere uses thread pools to manage concurrent tasks. The Maximum Size property of a thread pool can be set in the administrative console:

Servers → **Application servers** → ***server_name*** → **Additional Properties** → **Thread Pools** → ***thread_pool_name***

The following thread pools typically need to be tuned:

- Default
- ORB.thread.pool
- WebContainer

In addition, thread pools used by Work Managers are configured separately in the console via:

Resources → **Asynchronous beans** → **Work managers** → ***work_manager_name*** **> Thread pool properties**

The following Work Managers typically need to be tuned:

- DefaultWorkManager
- BPENavigationWorkManager

3.3.5 JMS connection pool sizes

There a few ways of accessing the JMS connection factories and JMS queue connection factories from the 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*** **(e.g. SIB JMS Resource Adapter)** → **Additional proerpties** → **J2C connection factories** → ***factory_name***

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

3.3.6 JDBC data source parameters

Data sources can be accessed from either of these paths:

- **Resources** → **JDBC** → **Data sources** → ***data_source_name***
- **Resources** → **JDBC Providers** → ***JDBC_provider_name*** → **Additional Properties** → **Data sources** → ***data_source_name***

Connection pool size

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 need to be tuned:

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

For WebSphere Process Server, the BPEDB data source should typically be tuned to a higher value; start by setting this to 300.

3.3.7 Messaging engine properties

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

- sib.msgstore.discardableDataBufferSize
  - In memory buffer for best effort nonpersistent messages.
  - Default is 320K.
  - Once full, messages will be discarded to allow newer messages to be written to the buffer.
- sib.msgstore.cachedDataBufferSizeCachedDataBufferSize
  - In memory cache for messages other than best effort nonpersistent
  - Default is 320K

These properties can be accessed in the console by selecting **Service Integration → Buses → *bus_name* → Messaging Engines → *messaging_engine_name* → Additional properties → Custom properties**.

Full details of these are given in the Information Center article at http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.pmc.doc/concepts/cjk_learning.html.

3.3.8 Run production servers in production mode

WebSphere application servers can be run in development mode to 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 since it is not designed to produce optimal runtime performance. Make sure the "Run in development mode" checkbox for the server is unchecked. This setting is found in the server's configuration panel in the administrative console. Select **Servers → Application Servers → *server_name* → Configuration**.

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 (CEI) 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: Therefore, it is recommended that tracing and monitoring be used judiciously and when possible, turned off entirely to ensure optimal performance.

Most tracing and monitoring is controlled through the administrative console. Please validate that the appropriate level of tracing and monitoring is set for the PMI Monitoring, Logging, and Tracing settings via the administrative console.

Further, use the administrative console to validate that the "Audit logging" and "Common Event Infrastructure logging" check boxes are disabled in the Business Flow Manager and the Human Task Manager, unless these capabilities are required for business reasons.

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

3.4.2 Tuning for large objects

In this section we discussing tuning for performance when using 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 big as possible on two commonly used platforms. For more comprehensive heap setting techniques, consult 3.4.13, “Advanced Java heap tuning” on page 54.

- Windows®

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

- AIX

  Using the normal Java heap settings, the Java5 and Java 6 JVM supports heaps 2 GB to 2.4 GB on 32-bit systems. Note that since the 4GB 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, etc. 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 needs to 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 (e.g. 10 seconds) and low value for pollQuantity (e.g. 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.

3.4.3 Tuning for maximum concurrency

For most high volume deployments on server-class hardware, there will be many operations which take place simultaneously. Tuning for maximum concurrency ensures that the server will accept enough load to saturate its core(s). One sign of an inadequately tuned configuration is when additional load does not result in additional core utilization, while the 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 needs to 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 business objects come from the adapter, ensure the adapter is tuned for concurrent delivery of input messages.

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

If the input business objects come from messaging, the ActivationSpec (MDB bindings) and Listener ports (MQ or MQJMS bindings) need to 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 up to 10 business objects from the JMS queue can be delivered to the MDB threads concurrently. Change it to 100 if a concurrency of 100 is desired.

Note that the Tivoli Performance Viewer 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 called UnavailableMessageCount that gives you the number of unavailable messages on each queue point (select ***resource_name*** → **SIB Service** → **SIB Messaging Engines** → **bus_name** → **Destinations** → **Queues**). If any queue has at least *maxConcurrency* unavailable messages, this would imply that the number of messages on the queue 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 thread pool sizes

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

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

Threads in the WebContainer 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 and it needs to be tuned, likely to a higher value than the default.

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

Configure dedicated thread pools for MDBs

The Default thread pool is shared by many WebSphere Application Server tasks. It is sometimes desirable to separate the execution of JMS MDBs to a dedicated thread pool. Follow the steps below to change the thread pool used for JMS MDB threads.

1. Create a new thread pool, say MDBThreadPool, on the server. **Servers** → **Server Types** → **WebSphere application servers** → **server** → **Thread pools**. Then, click **New**.

2. Open the service integration bus (SIB) JMS Resource Adapter administrative console panel with server scope from **Resources** → **Resource Adapters** → **Resource adapters**. If the adapter is not shown, go to Preferences, and set the Show built-in resources checkbox.
3. Change the thread pool alias from Default to MDBThreadPool.
4. Repeat the 2 and 3 for SIB JMS Resource Adapters at the node and cell scope.
5. Restart the server for the change to be effective.

SCA Module MDBs for asynchronous SCA calls use a separate resource adapter, the Platform Messaging Component SPI Resource Adapter. Follow the same 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 business object is handled by a single thread from end to end, the tuning for the edge components is normally adequate. In many situations, however, there are multiple thread switches during the end to end execution path. It is important to tune the system to ensure adequate concurrency for each asynchronous segment of the execution path.

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

The asynchrony in a long running business process occurs at transaction boundaries (see 2.5, "Transactionality considerations" on page 17 for more details on settings that affect transaction boundaries). BPE defines an internal MDB and its ActivationSpec: BPEInternalActivationSpec. The maxConcurrency parameter needs to be tuned following the same guideline as for 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 the underlying resource are blocked until the connections are freed up and allocated to them by the pool manager. If no connection is freed in the specified interval, a ConnectionWaitTimeout exception is issued.

If 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 maximum connection pool size needs to 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 configuration 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 and WebSphere ESB servers, e.g. the BPEDB data source, WPSDB data source, and the messaging engine database 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 300.

3.4.4 Messaging tuning

This section discusses performance tuning for messaging.

For messaging engines, choose data store or file store

Messaging engine persistence is usually backed by a database. Stating with the 6.2.0 release, a standalone configuration of WebSphere Process Server or WebSphere ESB can have the persistence storage of BPE and SCA buses backed by the file system (file store). The choice of file store has to be made at profile creation time. Use the Profile Management Tool to create a new standalone enterprise service bus profile or standalone process server profile. Choose **Profile Creation Options** → **Advanced profile creation** → **Database Configuration**, select checkbox **Use a file store for Messaging Engine (MEs**). When this profile is used, file stores 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 in the administrative console by selecting **Service Integration-Buses** → ***bus_name*** → **Messaging Engines** → ***messaging_engine_name*** → **Additional properties** → **Custom properties**.

Move messaging engine data stores to a high performance DBMS

For better performance, the messaging engine data stores 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 messaging engine data store, the method outlined in this section can be used to change the data store 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 messaging engine each. The example below shows the buses in WebSphere Process Server installed on machine box01; the node and cell names are the default.

Table 3-1 Service integration buses and messaging engines installed on 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 messaging engines is by default configured to use a data store in Derby. Each data store 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 data stores. The individual data stores/tables are completely separate and each messaging engine acquires an exclusive lock on its set of tables during startup. Each messaging engine uses a unique schema name to identify its set of tables.

Setting up the data stores for the messaging engines

For information on setting up the data stores see *Messaging engines and data stores* in the WebSphere Application Server V7 Information Center at
`http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.pmc.nd.multiplatform.doc/tasks/tjm0005_.html`.

Create the DB2 database and load the data store schemas

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

Table 3-2 Mesasging engine 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.

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

   In this example (used on a Windows operating system), *WAS Install* represents the WebSphere Process Server Installation directory and *user* represents the user name.

2. Repeat the command for each schema/messaging engine.
3. To be able to distribute the database across several disks, edit the created schema definitions and put each table in a tablespace named after the schema used i.e. 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 tablespace per data store was used.

4. After creating all schema definitions and defined tablespaces for the tables, create a database named SIB.
5. Create the tablespaces and distribute the containers across available disks by issuing the following command for a system managed tablespace:

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

   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.

Please see Sections 3.4.10, “Database: general tuning” on page 46 and 3.4.11, “Database: DB2 specific tuning” on page 48 for further information on database and DB2-specific tuning, respectively.

Create the data sources for the messaging engines

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

Table 3-3 shows the default state:

Table 3-3 JDBC provider defaults

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

Follow these steps:

1. Create a new JDBC provider of type DB2 Universal JDBC Driver Provider for the non-XA data sources if it doesn't already exist. The XA DB2 JDBC Driver Provider should exist if BPC was configured correctly for DB2.
2. Create four new JDBC data sources, one for CEI as an XA data source, the remaining three as single-phase commit (non-XA) data sources.

Table 3-4 provides new names.

Table 3-4 New data sources

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 checkbox named "Use this Data Source in container managed persistence (CMP)".
2. Set a component-managed authentication alias.
3. Set the database name to the name used for the database created earlier for messaging, e.g. SIB.
4. Select a driver type of type 2 or type 4. Per DB2 recommendations, use the JDBC Universal Driver type 2 connectivity to access local databases and type 4 connectivity to access remote databases.. Note that a driver of type 4 requires a host name and valid port to be configured for the database.

Change the data stores of the messaging engines

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

1. In the Navigation Panel go to **Service Integration** → **Buses** and change the data stores for each bus/messaging engine displayed.
2. Put in the new JNDI and schema name for each data store. Uncheck the checkbox "Create Tables" since the tables have been created already.
3. The server immediately restarts the messaging engine. The SystemOut.log shows the results of the change and also shows if the messaging engine starts successfully.
4. Restart the server and validate that all systems come up using the updated configuration.

The last remaining task is the tuning of the database; please see 3.4.10, “Database: general tuning” on page 46 and 3.4.11, “Database: DB2 specific tuning” on page 48 for further information on 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 the 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 administrative console to make these changes.

1. Set the 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 wildcard (*) for the host name of the Web Container port. Replace it with the hostname or IP address. The property can be accessed from **Application servers** → **messaging engine** → **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 hostname 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 (e.g. localhost:9080) in the URL.
4. Make sure there is not an entry for your server hostame 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 Business Process Choreographer.

Tuning Work-Manager-based navigation for business processes

Starting with WebSphere Process Server 7.0, work-manager-based navigation is the default navigation mode for WebSphere Process Server (versus JMS-based navigation). Work-Manager-based navigation provides two performance optimizations, keeping the quality of service of process navigation consistent with persistent messaging (JMS-based navigation):

- Work-Manager-based navigation. A WorkManager is a thread pool of J2EE threads. WorkManager process navigation exploits 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, a part of the Work-Manager-based navigation mode which 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 are found in the administrative console by navigating 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 Work-Manager-based navigation and InterTransactionCache optimizations.
- Work-Manager-Based Navigation Message Pool Size: this property specifies the size of the cache used for navigation messages that cannot be processed immediately, provided Work-Manager-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, WebSphere Process Server uses JMS-based navigation for new messages, so for optimal performance ensure this Message Pools size is set to a sufficiently high value.
- 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 using These settings can be found by selecting **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 35. If the 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.

Tuning the business process container for JMS navigation

If JMS-based navigation is configured, the following resources need to be optimized for efficient navigation of business processes:

- 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 in the administrative console by selecting **Resources** → **Activation Specifications** → **BPEInternalActivationSpec**.

- JMS connection factory BPECFC

  Set the connection pool size to:

  ```
  Number of threads in the BPEInternalActivationSpec + 10%
  ```

  This resource can be found in the administrative console at **Resources** → **JMS** → **Connection factories** → **BPECFC** → **Connection pool properties**. Note that this connection factory is also used when work-manager based navigation is in use, but only for error situations or if the server is highly overloaded.

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 improve query performance with potentially some impact on process navigation performance. For DB2, the DB2 design advisor can be used to guide in choosing indexes.

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, or 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:

- API calls for task list and process list queries may take more time to respond, 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 load and to utilize the CPU. However, increasing the parallelism of API call execution beyond what is necessary can negatively influence response times. Also, increased parallelism can put excessive load on the BPC database. When tuning the parallelism of API calls, measure response times before and after tuning, and adjust the parallelism if necessary.

3.4.7 WebSphere ESB tuning

Following are additional configuration options that are relevant to tuning WebSphere ESB. Please see 4.2, “WebSphere ESB settings” on page 64 for a suggested set of initial values to use.

Tune the database if using persistent messaging

If you are using persistent messaging the configuration of your database becomes important. Use a remote DB2 instance with a fast disk array as the database server. You may also find benefit in tuning the connection pooling and statement cache of the data source. Please see sections 3.4.10, “Database: general tuning” on page 46 and 3.4.11, “Database: DB2 specific tuning” on page 48 for further information on tuning DB2, and also note the relevant references in “Related publications” on page 67.

Disable event distribution for CEI

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

Event distribution can be disabled from the administrative console by selecting **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 (e.g. using the endpoint lookup mediation primitive), results from the registry are cached in WebSphere ESB. The lifetime of the cached entries can be configured from the administrative console by selecting **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 will result 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 Clustered topology tuning

One 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 WebSphere Process Server Network Deployment (ND) infrastructure provides this capability. However, effective scaling still requires standard performance monitoring and bottleneck analysis techniques to be used.

Here are some considerations, and tuning guidelines, when expanding or tuning a clustered topology. In the discussion below, we assume additional cluster members also 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 (core, disk, network, etc) 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 messaging engine cluster is servicing multiple busses, and messaging traffic is spread across these busses, consider employing a separate messaging engine 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. Partitioning a bus also has considerations for balancing work across the messaging engine cluster members. For further information, please see the following:

      `http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.pmc.nd.doc/concepts/cjt0014_.html`
  - A database server may become the bottleneck. Approaches to consider are:
    - If the database server is hosting multiple databases that are active (for example, the BPEDB and the MEDB), consider hosting each database on a separate server.
    - If a single database is driving load, consider a more powerful database server.
    - Beyond these items, database partitioning and clustering capabilities can be exploited.

3.4.9 WebSphere Business Monitor tuning

This section provides advanced tuning recommendations for WebSphere Business Monitor.

Configure Java heap sizes

The default maximum heap size in most implementations of Java is 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.

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. Performance is improved by avoiding this double-store, which can be done using the 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

Beyond its persistent delivery of events to registered consumers, CEI offers the ability to explicitly store events in a database. This has significant performance overhead and should be avoided if this additional functionality is not needed. The CEI Data Store is also configured in the administrative console:

Service Integration → Common Event Infrastructure → Event Service → Event Services → Default Common Event Infrastructure event server: deselect **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 have to roll back. If you experience frequent rollbacks, consider reducing the batch size. This can be done in the administrative console under the server scope. Select:

Applications → Monitor Models → *version* → Runtime Configuration → Tuning → Message Consumption Batch size: <default 100>

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 in the WebSphere administrative console. Select:

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 provides general tuning hints for databases.

Provide adequate statistics for optimization

Databases often have a wide variety of available choices when determining the best approach to accessing data. Statistics, which describe the “shape” of the data, are used to guide the selection of a low-cost data access strategy. 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.

Gathering statistics can be expensive, but fortunately for many workloads a set of representative statistics will allow good performance over a large span of time. It may be necessary to refresh statistics periodically if the data population shifts dramatically.

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 buffer fills, at transaction commit time, and for some implementations after a maximum interval of time. As a result, database log files may be heavily utilized. More importantly, log writes 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 a separate device from the tablespace containers

A basic strategy for all database storage configurations is to place the database logs on dedicated physical disks, ideally on a dedicated disk adapter. 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 of the log stream. Such separation also improves recoverability when log archival is employed.

Provide sufficient physical memory

Accessing data in memory is of course much faster than reading it from disk. With 64-bit hardware being readily available and memory prices continuing to fall, for many performance critical workloads it makes sense to provision enough memory to avoid most disk reads in steady state.

Great care should be taken to avoid virtual memory paging in the database machine. The database manages its memory with the assumption that it is never paged, and does not cooperate well should the operating system decide to swap some of its pages to disk.

Avoid double buffering

Since the database attempts to keep frequently accessed data in memory, in most cases there is no benefit to using file system caching. In fact, performance typically improves by using direct I/O, when files read by the database bypass the file system cache and only one copy of the data is held in memory. This allows more memory to be given to the database and avoids overheads in the file system as it manages its cache.

A further advantage can be gained on some operating systems such as AIX by using concurrent I/O. This bypasses per-file locking, shifting responsibility for concurrency control to the database and in some cases allowing more useful work to be offered to the adapter or the device.

An important exception to this guideline occurs for large objects (LOB, BLOB, CLOB, etc.) which are not buffered by the database itself. In this case it can be advantageous to arrange for file system caching, preferably only for files which back large objects.

Refine table indexes as required

WebSphere BPM products typically provide a reasonable set of indexes for the database tables they use. In general, creating indexes involves a tradeoff between the cost of queries and the cost of statements which insert, update, or delete data. For query intensive workloads, it makes sense to provide a rich variety of indexes as required to allow rapid access to data. For update intensive workloads, it is often helpful to minimize the number of indexes defined, as each row modification may require changes to multiple indexes. Note that indexes are kept current even when they are infrequently used.

Index design therefore involves compromises. The default set of indexes may not be optimal for the database traffic generated by a BPM product in a specific situation. If database CPU or disk utilization is high or there are concerns with database response time, it may be helpful to consider changes to indexes.

DB2 and Oracle databases provide assistance in this area by analyzing indexes in the context of a given workload. Recommendations are given to add, modify, or remove indexes. One caveat is that if the workload does not capture all relevant database activity then a necessary index might appear unused, leading to a recommendation that it be dropped. If the index is not present, future database activity could suffer as a result.

3.4.11 Database: DB2 specific tuning

Providing a comprehensive DB2 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 DB2 environments. In the following sections, we discuss these rules, and provide pointers to more detailed information. The complete set of current DB2 manuals (including database tuning guidelines) can be found by using the DB2 Information Center:

http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp

Another excellent reference is *Best practices for DB2 for Linux®, UNIX®, and Windows*, available at:

http://www.ibm.com/developerworks/data/bestpractices/

Update database statistics

DB2 provides an Automatic Table Maintenance feature, which runs the RUNSTATS command in the background as required to ensure that the correct statistics are collected and maintained. This is controlled by the database configuration parameter auto_runstats, and is enabled by default for databases created by DB2 V9.1 and beyond. See also the "Configure Automatic Maintenance..." wizard at the database level in the DB2 Control Center.

One approach to manually updating statistics on all tables in the database is to use the REORGCHK command. Dynamic SQL, such as that produced by JDBC, will immediately take the new statistics into account. Static SQL, like that in stored procedures, must be explicitly rebound in the context of the new statistics. Here is an example which performs these steps to gather basic statistics 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.

It is generally better to gather additional statistics, so consider also using the following command for every table requiring attention:

```
runstats on table <schema>.<table> with distribution and detailed indexes
```

Set buffer pool sizes correctly

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

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

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.

DB2 V9.1 and beyond provide self tuning memory management, which includes managing buffer pool sizes. This is controlled globally by the self_tuning_mem database level parameter, which is ON by default. Individual buffer pools can be enabled for self tuning using SIZE AUTOMATIC at CREATE or ALTER time.

To choose appropriate buffer pool size settings manually, 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

The DB2 Design Advisor, available from the Control Center, provides recommendations for schema changes, including changes to indexes. It can be launched from the menu presented when right-clicking on a database in the left column.

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 4K pages in each log file. “logprimary” controls the number of primary log files. The Control Center also provides a “Configure Database Logging...” wizard.

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 gives additional opportunities for write coalescing at the cost of longer recovery time.

The default value softmax is 100, meaning that the database manager will attempt to clean pages such that a single log file needs to be processed during recovery. For best performance, we recommend increasing this to 300, meaning that 3 log files may need processing during recovery:

```
db2 update db config for yourDatabaseName using softmax 300
```

Use SMS for tablespaces containing large objects

When creating REGULAR or LARGE tablespaces in DB2 V9.5 and beyond which contain performance critical LOB data, we recommend specifying MANAGED BY SYSTEM to gain the advantages of cached LOB handling in SMS.

Among WebSphere BPM products, this consideration applies to:

- WebSphere Process Server: the Process Choreagrapher database, sometimes called BPEDB.
- WebSphere Process Server and WebSphere ESB: databases backing service integration bus messaging engine data stores.

For background, see “Avoid double buffering” on page 47. A detailed explanation follows.

DB2 tablespaces can be configured with NO FILE SYSTEM CACHING, which in many cases improves system performance.

If a tablespace is MANAGED BY SYSTEM, then it uses System Managed Storage (SMS) which provides desirable special case handling for LOB data with regard to caching. Even if NO FILE SYSTEM CACHING is in effect (by default or as specified) access to LOB data still uses the file system cache.

If a tablespace is MANAGED BY DATABASE, then it uses Database Managed Storage (DMS) which does not differentiate between LOB and non-LOB data with regard to caching. In particular, NO FILE SYSTEM CACHING means that LOB access will be directly to disk for both reads and writes. Unconditionally reading LOBs from disk can cause high disk utilization and poor database performance.

Since V9.1, DB2 has by default created databases which use automatic storage (AUTOMATIC STORAGE YES), meaning that the database manages disk space allocations itself from one or more pools of available file system space called storage paths. If automatic storage is enabled, CREATE TABLESPACE will use it by default (MANAGED BY AUTOMATIC

STORAGE). For non-temporary tablespaces, REGULAR and LARGE, automatic storage is implemented using DMS on files.

Before DB2 V9.5 the default caching strategy for tablespaces was FILE SYSTEM CACHING. In V9.5, this was changed to NO FILE SYSTEM CACHING for platforms where direct I/O or concurrent I/O is available. Taking defaults on V9.5 we now have a database with AUTOMATIC STORAGE YES, and a tablespace which is MANAGED BY AUTOMATIC STORAGE and in many cases NO FILE SYSTEM CACHING. Such a tablespace, which is implemented using DMS, will not cache LOBs in the buffer pool or the file system.

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 4K 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 which 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 which lock many rows into smaller transactions. It is usually simpler to try tuning the database first.

Beginning with Version 9, DB2 adjusts the locklist and maxlocks parameters automatically by default. To manually tune these, 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 impacts 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.

Bound the size of the catalog cache for clustered applications

The catalog cache is used to avoid repeating expensive activities, notably preparing execution plans for dynamic SQL. Thus it is important that the cache be sized appropriately.

By default, several 4 KB pages of memory are allocated for each possible application as defined by the MAXAPPLS database parameter. The multiplier is 4 for DB2 9, and 5 for DB2

9.5 and beyond. MAXAPPLS is AUTOMATIC by default, and its value is adjusted to roughly match the peak number of applications connected at runtime.

When running clustered applications, such as those deployed in the Process Choreographer in WebSphere Process Server, we have observed a value of more than 1000 for MAXAPPLS, meaning that at least 4000 pages would be allocated for the catalog cache given default tuning. For the same workload, 500 pages were sufficient:

```
db2 update db config for yourDatabaseName using catalogcache_sz 500
```

The default behavior assumes heterogeneous use of database connections. A clustered application will typically have more homogeneous use across connections, allowing a smaller package cache to be effective. Bounding the package cache size frees up memory for other more valuable uses.

To manually tune the CATALOGCACHE_SZ database parameter, see the recommendations documented at:

http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.udb.admin.doc/doc/r0000338.htm

(Before DB2 V9.5) size the database heap appropriately

DB2 Version 9.5 and beyond provide AUTOMATIC tuning of the database heap by default. We recommend using this when available.

To manually tune the DBHEAP database parameter, see the recommendations documented here:

http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.udb.admin.doc/doc/r0000276.htm

(Before DB2 V9.7) size the log buffer appropriately

Before DB2 Version 9.7 the default LOGBUFSZ is only 8 pages. We recommend setting this to 256, which is the default in Version 9.7:

```
db2 update db config for yourDatabaseName using logbufsz 256
```

(DB2 V9.7 and beyond) consider disabling current commit

DB2 Version 9.7 supports new query semantics which always return the committed value of the data at the time the query is submitted. This support is ON by default for newly created databases. We found that performance improved in some cases when we disabled the new behavior, reverting to the original DB2 query semantics:

```
db2 update db config for yourDatabaseName using cur_commit disabled
```

Recommendations for WebSphere Process Server

The following link discussess 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/v7r0mx/index.jsp?topic=/com.ibm.websphere.bpc.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 gives details on BPEDB database creation, including pointers to useful creation scripts for a production environment.

http://publib.boulder.ibm.com/infocenter/dmndhelp/v7r0mx/index.jsp?topic=/com.ibm.websphere.bpc.doc/doc/bpc/t2codbdb.html

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 when used with WebSphere BPM products. In the sections below, we discuss these rules, and provide pointers to more detailed information. In addition, the following references are useful:

- Oracle Database 11g Release 1 documentation (includes a Performance Tuning Guide):

  http://www.oracle.com/pls/db111/homepage
- A white paper discussing Oracle on AIX:

  http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100883

Update database statistics

Oracle provides an automatic statistics gathering facility, which is enabled by default. One approach to manually updating statistics on all tables in a schema is to use the dbms_stats utility; please consult the Oracle product documentation for further information.

Set buffer cache sizes correctly

Oracle provides automatic memory management for buffer caches. For additional discussion on configuring automatic memory management and for guidance on manually setting buffer cache sizes, please see the following references.

- For Oracle 10g R2:

  http://download.oracle.com/docs/cd/B19306_01/server.102/b14211/memory.htm#i29118
- For Oracle 11g R1:

  http://download.oracle.com/docs/cd/B28359_01/server.111/b28274/memory.htm#i29118

Maintain proper table indexing

The SQL Access Advisor, available from the Enterprise Manager, provides recommendations for schema changes, including changes to indexes. It can be found starting at the database home page, then following the Advisor Central link in the Related Links section at the bottom of the page.

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 which generate a high volume of log traffic need larger log files to achieve this goal.

Recommendations for WebSphere Process Server

The following link discusses specifying initial Oracle database settings:

http://publib.boulder.ibm.com/infocenter/dmndhelp/v7r0mx/index.jsp?topic=/com.ibm.websphere.bpc.doc/doc/bpc/t5tuneint_spec_init_db_oracle.html

The following link discusses creating an Oracle database for Business Process Choreographer and gives details on BPEDB database creation, including pointers to useful creation scripts for a production environment.

http://publib.boulder.ibm.com/infocenter/dmndhelp/v7r0mx/index.jsp?topic=/com.ibm.websphere.bpc.doc/doc/bpc/t2codbdb.html

The default Oracle policy for large objects (LOB) is to store the data within the row, when the size of the object does not exceed a threshold. In some cases, workloads have LOBs which regularly exceed this threshold. By default, such LOB accesses bypass the buffer cache, meaning that LOB reads are exposed to disk I/O latencies when using the preferred direct or concurrent path to storage.

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 authoring and runtime performance. The most important of these are related to garbage collection and setting the Java heap size. This section will deal with these topics in detail.

Note that the products covered in this paper utilize IBM JVMs on most platforms (AIX, Linux, Windows, etc.), and the HotSpot JVMs on selected other systems, such as Solaris and HP/UX. Vendor specific JVM implementation details and settings will be discussed as appropriate. Also note that all BPM v7 products in this document use Java 6. It has characteristics similar to Java 5 used in the BPM v 6.1 and v6.2.0 products, but much different from Java 1.4.2 used by V6.0.2.x and earlier releases. For brevity, only Java 6 tuning is discussed here.

Following is a link to the IBM Java 6 Diagnostics Guide:

http://publib.boulder.ibm.com/infocenter/javasdk/v6r0/index.jsp

The guide 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 6 garbage collection algorithms, please see Section "Memory Management" in the chapter titled "Understanding the IBM SDK for Java."

Sun HotSpot JVM references follow:

- The following URL provides a useful summary of HotSpot JVM options for Solaris:

  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 of Sun's HotSpot JVM, follow the URL below.

  http://java.sun.com/docs/performance/

Monitoring garbage collection (GC)

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 3-2 Example 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 3-3 Example Solaris HotSpot JVM verbosgc trace output (young and old)

```
[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® through this URL:

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

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

Setting the heap size for most configurations

This section contains guidelines for determining the appropriate Java heap size for most configurations. If your configuration requires that more than one JVM runs concurrently on the same system (for example, if you run both WebSphere Process Server and WebSphere Integration Developer on the same system), then you should also read the next section, “Setting the Heap Size when running multiple JVMs on one system” on page 56. If your objective is to support large business objects, read “Tuning for large objects” on page 34.

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.

There are several approaches to setting optimal heap sizes. We describe here the approach that most applications should use when running the IBM JVM on AIX. The essentials can be applied to other systems. Set the initial heap size (-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 be 1/2 of the heap size or larger, and it should not be capped at 64MB. New area sizes are set by JVM options: -Xmn<size>, -Xmns<initialSize>, and -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 between 1/4th and 1/2 of the heap size. Note that you should never increase the nursery to more than 1/2 the full heap. The nursery size is set using the MaxNewSize and NewSize parameters (that is, -XX:MaxNewSize=128m, -XX:NewSize=128m).

After the heap sizes are set, 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 is could be a memory over-usage problem. Conversely, if response times are too long due to GC pause times, decrease the heap size. If both problems are observed, an analysis of the application heap usage is required.

Setting the Heap Size when running multiple JVMs on one system

Each running Java program has a heap associated with it. Therefore, if you have a configuration where more than one Java program is running on a single physical system, setting the heap sizes appropriately is of particular importance. An example of one such configuration is when the WebSphere Integration Developer is on the same physical system as WebSphere Process Server. Each of these is a separate Java program that has its own Java heap. If the sum of all of the virtual memory usage (including both Java Heaps as well as all other virtual memory allocations) exceeds the size of physical memory, the Java heaps will be subject to paging. As previously noted, this causes total system performance to degrade significantly. To minimize the possibility of this occurring, use the following guidelines:

- Collect a verbosegc trace for each running JVM.
- Based on the verbosegc trace output, set the initial heap size to a relatively low value. For example, assume that the verbosegc trace output shows that the heap size grows quickly to 256 MB, and then grows more slowly to 400 MB and stabilizes at that point. Based on this, set the initial heap size to 256 MB (-Xms256m).
- Based on the verbosegc trace output, set the maximum heap size appropriately. Care must be taken to not set this value too low, or 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 (e.g. if a period of peak activity drives a

higher throughput rate), so setting the maximum heap size somewhat higher than current requirements is generally a good policy.

- Be careful to not set the heap sizes too low, or garbage collections will occur frequently, which might reduce throughput. Again, a verbosegc trace 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 OutOfMemory errors occur

The java.lang.OutOfMemory exception is used by the JVM in a variety of circumstances, making it sometimes difficult to track down the source of the exception. There is no conclusive mechanism for telling the difference between these potential error sources, but a good start is to collect a trace using verbosegc. If the problem is a lack of memory in the heap, then this is easily seen in this output. Please see "Monitoring garbage collection (GC)" on page 54 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 you 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 if the finalizers can be avoided, or minimized.

Note that OutOfMemory errors can also occur for issues unrelated to JVM heap usage, such as running out of certain system resources. Examples of this include insufficient file handles or thread stack sizes that are too small.

In some cases, you can tune the configuration to avoid running out of native heap: try reducing the stack size for threads (the -Xss parameter). However, deeply nested methods may force a thread stack overflow if there is insufficient stack size.

For middleware products, if you are using an in-process version of the JDBC driver, it is usually possible to find an out-of-process driver that can have a significant effect on the native memory requirements. For example, you can use type 4 JDBC drivers (DB2's Net drivers, 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 environment variable settings 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.

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

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 whan a system is under high load, so consider whether or not 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 WebSphere Process Server tuning for WebSphere InterChange Server migrated workloads

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

- For JMS based messaging used to communicate with legacy WBI adapters or custom adapters , make use of non-persistent queues when possible.
- For JMS based messaging used to communicate with legacy 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 service integration bus destinations via MQ Link. MQ Link is a protocol translation layer which 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.
- Turn off server logs for verbose workloads. Some workloads emit log entries for every transaction thus causing constant disk writes reducing overall throughput. Explore the possibility of turning off server logs to reduce the throughput degradation for such workloads.

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 28, 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 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 27. 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 28.

We discuss two settings in this section:

- A three-tier setup with the BPE database located on a separate server
- A two-tier (client/server) setup with the BPE database co-located on the server

4.1.1 Three tier configuration with Web services and a remote DB2 system

A three-tier configuration was used in our internal performance work to evaluate the performance of a business process that models automobile insurance claims processing. This configuration is an example of many production environments, where DB2 is on a separate system than WebSphere Process Server. 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 microflow plus macroflow pattern, where the macroflow is invoked 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 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:

The following settings and configuration options are recommended for all topologies in this section:

- Use the production template
- Define the Common database as local DB2 type 4
- Establish Business Process support with bpeconfig.jacl (note that this sets the Data sources → BPEDataSourceDb2 → WebSphere Application Server data source properties statement cache to 300)
- Disable PMI
- Set HTTP maxPersistentRequests to -1
- Set GC policy to –Xgcpolicy:gencon (see Table 4-1 on page 61 and Table 4-2 on page 61 for nursery setting –Xmn)
- Use remote DB2 databases (connection type 4) for BPE, SIB System, and SIB BPC databases

Table 4-1 shows WebSphere Process Server related settings to modify from their default value when WebSphere Process Server is deployed on AIX.

Table 4-1 Three tier application cluster settings for AIX

Setting	Value
Java Heap Megabytes	1536
Java nursery Megabytes –Xmn	768
Default Thread Pool Max	100
BPEDB Data source → connection pool max	300
BPEDB Data source → WebSphere Application Server data source properties → Statement cache size	300
BPC messaging engine data source → connection pool max	50
SCA SYSTEM messaging engine data source → connection pool max	50
WebSphere Process Server Common Data source → connection pool max	500
J2C activation specifications → SOABenchBPELMod2_AS → Custom properties → maxConcurrency, maxBatchSize	50,
Resources → Asynchronous Beans → Work Managers → BPENavigationWorkManager → Work request queue size, max threads, growable	400, 50, no
Application Cluster → Business Flow Manager → Message pool size, Intertransaction cache size	5000, 400
WebContainer Thread Pool Min,Max	100, 100
com.ibm.websphere.webservices.http.maxConnection	50

Table 4-2 shows WebSphere Process Server related settings to modify from their default value when WebSphere Process Server is deployed on Windows and Linux on Intel systems.

Table 4-2 Three tier Web service and remote DB2 tuning variations Windows and Linux on Intel systems

Tuning Variations	Microflow - Number of Cores			Macroflow - Number of Cores	
	1	2	4	1	4
Java Heap Megabytes	1280	1280	1280	1280	1280
Java nursery Megabytes -Xmn	640	640	640	768	768
Web Container Thread Pool Max	100	150	150	100	300
Default Thread Pool Max	100	200	200	100	200
BPE database connection pool max	150	250	250	150	350
BPC messaging engine database connection pool max	30	30	30	30	150
SYSTEM messaging engine database connection pool max	30	40	40	30	100

Common database connection pool max	80	80	80	80	100
J2C activation specifications → SOABenchBPELMod2_AS → Custom properties → maxConcurrency	40	40	40	160	160
BPEInternalActivationSpec batch size				10	10
SOABenchBPELMod2_AS batch size				32	32
Java custom property com.ibm.websphere.webservices.http.maxConnection	100	200	200	200	200
Application servers → server1 → Business Flow Manager → allowPerformanceOptimizations				Yes	Yes
Application servers → server1 → Business Flow Manager → interTransactionCache.size				400	400
Application servers → server1 → Business Flow Manager → workManagerNavigation.messagePoolSize				4000	4000
Resources → Asynchronous Beans → Work Managers → BPENavigationWorkManager → min threads, max threads, request queue size				30, 30, 30	30, 30, 30

The DB2 database server has three databases defined for use by the WebSphere Process Server server. The database logs and tablespaces should be spread across a RAID array to distribute disk utilization. The SCA.SYSTEM.*cellname*.BUS database and the BPE database should be tuned as follows.

The SCA.SYSTEM.cellname.BUS database should be tuned as follows:

```
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 should be created and tuned as follows:

1. `db2 CREATE DATABASE bpedb ON /raid  USING CODESET UTF-8 TERRITORY en-us`
2. `db2 update db cfg for bpedb using logbufsz 512 logfilsiz 10000 logprimary 20 logsecond 10 auto_runstats off`
3. Using the WebSphere Process Server generated script: `db2 -tf createTablespace.sql`
4. Using the WebSphere Process Server generated script: `db2 -tf createSchema.sql`
5. `db2 alter bufferpool ibmdefaultbp size 132000`
6. `db2 alter bufferpool bpebp8k size 132000`

4.1.2 Two tier configuration using file store for JMS

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. A two tier configuration is used, with WebSphere Process Server and DB2 on the same physical system, as is common for Proof of Concepts when limited hardware is available. This configuration is not a representative production configuration, however. 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 4-3 shows the parameter settings recommended for this configuration.

Table 4-3 Two-tier configuration using JMS file store parameter settings

Parameter names	BPEDB 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
LOGSECOND	10
MAXAPPLS	90
MAXLOCKS	57
MINCOMMIT	1
NUM_IOCLEANERS	6
NUM_IOSERVERS	10
PCKCACHESZ	915
SOFTMAX	440

Parameter names	BPEDB settings
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 should also be 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 may need to be increased beyond the default of 50 threads.
- The maximum concurrency is set to 50 threads for activation specifications.
- The database connection pool size for the BPEDB is increased to 60, and the statement cache size for the BPEDB is increased to 300.
- The Maximum connections property for JMS connection pools is set to 40.
- Connectivity to the local database uses the DB2 JDBC Universal Driver Type 2 driver. Type 2 drivers produce better performance when WebSphere Process Server and DB2 are on the same physical system.
- WebSphere Process Server JVM heap size is set to a fixed size of 1024 MB. In addition, the gencon garbage collection policy is used.

4.2 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 27. 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 28 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
- If security is required, use Application security instead of Java2 security to reduce overhead.
- Java Heap size is fixed at 1280 MB for Windows and 1280 MB for AIX
- Gencon garbage collection policy enabled (-Xgcpolicy:gencon), 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 JMS

The WebSphere ESB settings for MQ and JMS are:

- Activation specification - set maximum concurrent endpoints to 50
- Queue Connection factory - set the maximum connection pool size to 51
- DiscardableDataBufferSize set to 10MB and CachedDataBufferSize set to 40MB

4.2.4 DB2 settings for JMS persistent

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 Separate Device from Table Spaces
- Set Buffer Pool Size Correctly
- Set the Connection Min and Max to 30
- Set the Statement cache size to 40
- Raw partition for DB2 logs

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

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

For information about ordering these publications, see “How to get Redbooks” on page 68. Note that some of the documents referenced here may be available in softcopy only.

- *WebSphere Business Process Management V7 Production Topologies*, SG24-7854
- *Migrating WebSphere InterChange Server and Adapters to WebSphere Process Server V6.2 and Best Practices*, SG24-7415
- *WebSphere Business Process Management 6.2.0 Performance Tuning*, REDP-4551

Online resources

These Web sites are also relevant as further information sources:

- WebSphere BPM Version 7.0 information center

  http://publib.boulder.ibm.com/infocenter/dmndhelp/v7r0mx/index.jsp
- WebSphere Application Server Performance URL

  http://www.ibm.com/software/webservers/appserv/was/performance.html
- WebSphere Application Server 7.0 Information Center (including Tuning Guide)

  http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.base.doc/info/aes/ae/welcome_base.html
- Setting up a Data Store in the Messaging Engine

  http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.pmc.nd.multiplatform.doc/tasks/tjm0005_.html
- DB2 Best Practices for Linux, UNIX, and Windows

  http://www.ibm.com/developerworks/data/bestpractices/?&S_TACT=105AGX11&S_CMP=FP
- DB2 Version 9.7 Info Center

  http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp
- DB2 Version 9.5 Info Center

  http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/index.jsp
- Using JCA Adapters with WebSphere Process Server and WebSphere ESB

  http://www-128.ibm.com/developerworks/library/ws-soa-j2caadapter/index.html?ca=drs-
- WebSphere Process Server Support

  http://www-306.ibm.com/software/integration/wps/support/
- WebSphere ESB Support

http://www-306.ibm.com/software/integration/wsesb/support/

- IBM Java 6.0 Diagnostic Guide

  http://publib.boulder.ibm.com/infocenter/javasdk/v6r0/index.jsp
- Oracle Database 11g Release 1 documentation (includes a Performance Tuning Guide):

  http://www.oracle.com/pls/db111/homepage
- Oracle Architecture and Tuning on AIX v1.31:

  http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100883
- Oracle 10g Release 2 documentation (includes a Performance Tuning Guide)

  http://www.oracle.com/pls/db102/homepage

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

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

WebSphere Business Process Management (BPM) V7 Performance

Learn valuable tips for tuning

Get the latest best practices

Try the example settings

This IBM® Redpaper™ publication was produced by the WebSphere® Business Process Management performance teams. It provides performance tuning tips and best practices for WebSphere Process Server 7.0.0.1, WebSphere Enterprise Service Bus 7.0.0.1, WebSphere Integration Developer 7.0.0.1, WebSphere Business Monitor 7.0.0.0, and WebSphere Business Modeler 7.0.0.1. 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).

We envision this paper to be read by a wide variety of groups, both within IBM (development, services, technical sales, etc.) and by customers. For those who are either considering or are in the very early stages of implementing a solution incorporating these products, this document should prove a useful reference, both in terms of best practices during application development and deployment, and as a reference for setup, tuning and configuration information. This paper provides a useful introduction to many of the issues influencing each product's performance, and can serve as a guide for making rational first choices in terms of configuration and performance settings. Similarly, those who have already implemented a solution utilizing these products might effectively use the information presented here to gain insight as to how their overall integrated solution performance may be improved.

All of these products build on the core capabilities of the WebSphere Application Server infrastructure, so BPM solutions also benefit from tuning, configuration, and best practices information for WebSphere Application Server and the corresponding platform Java™ Virtual Machines (JVMs). Pointers to this information can be found in the Related publications. The reader is encouraged to use this paper in conjunction with these references.

REDP-4664-00