ORACLE

FUSION MIDDLEWARE

BUSINESS PROCESS MANAGEMENT

An Oracle White Paper
July 2011

Performance Tuning for Oracle Business Process Management Suite 10g

ORACLE

Introduction 1
Understanding the Goal 2
Typical Bottlenecks 2
Where to Start 3
Environmental Considerations 3
Use Current Versions 3
Use a J2EE Application Server and Clustering 4
Deploy Adequate Hardware 4
Application Design Considerations 5
Instance Size 5
Different Types of Variables 5
Use of Sub-processes 6
Handling Large Volumes 6
Tuning the BPM Engine 7
Audit Events Recommendations 7
Data Source Configuration 8
BPM 'To Do' JMS Queue Configuration 8
Enable "call by reference", if needed 8
JMS – Due Items Polling Interval Recommendations 9
Tuning the Directory Service 9
Master Group 9
LDAP Queries 10
Tracing 10
Tuning the PAPI Instance Cache 10
Important point about PAPI cache 11
Participant cache 11
Cache ProcesService object 11
Testing 12
Functional Testing 12
Cluster Testing 13

Load Testing 13
JVM Tuning 15
Recommended Settings 15
Heap Size 17
WebLogic Server Considerations 17
Non-persistent JMS Store 17
Transaction Logs 17
Database Tuning 18
Automatic Workload Repository 18
Missing Indexes 18
Use Separate Tablespaces for BLOBs 18
Conclusion 19

Introduction

Many organizations have built and deployed mission critical systems using Oracle Business Process Management Suite 10g. Other organizations are building such systems now. For these organizations considerations like the performance, scalability and reliability of their systems are paramount.

This white paper presents a set of tried and tested performance tuning best practices collected from successful deployments at many organizations over an extended period of time. While every application is different and has its own set of performance tuning challenges, we attempt to present a set of guidelines and a common sense approach that we hope will be broadly applicable.

Performance means different things to different people. For some it is a fast response time for users, for others it is the volume of work that can be processed within a given time period, for others still it is how rapidly a system can recover from a failure. The best practices collected in this white paper cover a broad spectrum of use cases. It is expected that several of them will be relevant in any given scenario.

Understanding the Goal

In order to maximize performance, you will need to monitor, analyze and tune all of the components that make up your application. This whitepaper describes the 'knobs' you can adjust and when and how you might want to adjust them.

To be effective, performance tuning needs to be comprehensive, iterative and address several levels:

- Configuration of the BPM software,
- The design of your application,
- Tuning of the application server,
- Tuning of the Java Virtual Machine,
- Tuning of the database,
- Tuning of operating system and kernel parameters,
- Tuning of disk and network I/O.

It is also important to understand that performance tuning is an iterative process. You need to make a small adjustment, then measure the impact, and then perform analysis and make another adjustment, and so on. Due to the vast differences in terms of applications the customers build using Oracle Business Process Management 10g, there are no global solutions that work well in every environment. Improving performance is a process of learning and testing.

Typical Bottlenecks

Experience shows that there are some common areas where bottlenecks tend to occur. Being aware of these typical bottlenecks can help us to focus our initial efforts in the areas where they are most likely to deliver the greatest return.

Mostly Automatic Processes

In environments that are characterized by mostly automatic processes, i.e. processes that do not involve human interaction through the BPM Workspace, the most common place to find performance problems, or bottlenecks, are the BPM Engine database and backend system latency.

In these cases you should start by tuning the BPM Engine, BPM Application Server and Engine database and then carefully watch backend system latency.

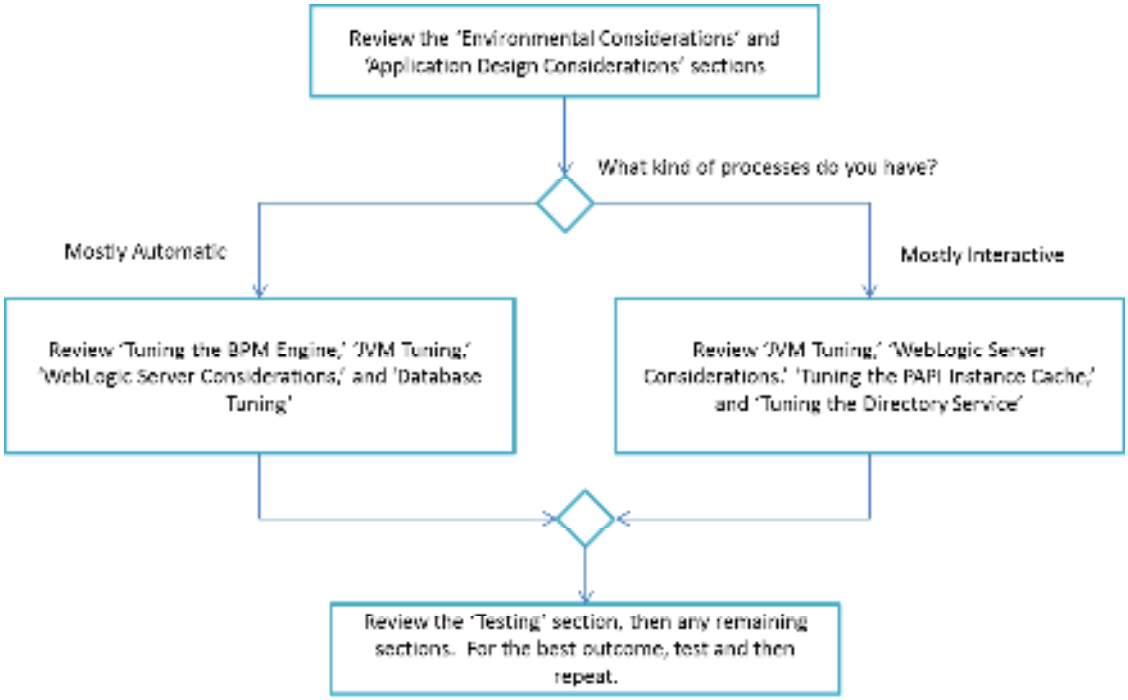
Mostly Interactive Processes

In environments that are characterized by mostly interactive processes, processes that do involve human interaction through the BPM Workspace, the most common bottleneck is the Workspace itself.

In these cases you should start by tuning the BPM Application Server and PAPI Instance Cache.

Where to Start

The flowchart below provides an overview of what order you might want to review the contents of this document, in order to get the most benefit with the least amount of work. Of course, it is not necessary to follow the flowchart. You may wish to read through the entire document and then select your own approach.



Environmental Considerations

Many performance issues can be avoided by ensuring that you have a well-designed environment. Prevention is better than cure!

Use Current Versions

You should build your environment using current versions of software unless there is a good reason not to do so. Using the current software versions ensures that you get the advantage of any patches that Oracle has released for issues encountered by other customers.

Practically speaking, this means you should aim to use the latest available production release in the 10g release stream, plus the latest available recommended cumulative patch set.

You should deploy patches into a non-production environment and test your application before rolling patches into production.

Use a J2EE Application Server and Clustering

You should run the BPM Enterprise server on a J2EE Application Server in preference to running it in Standalone mode. We recommend WebLogic Server. The recommendations in this whitepaper assume you are running BPM on WebLogic Server. If you are using a different application server, some recommendations may not apply.

If your application has any availability or performance requirements you should use a cluster in preference to using a single server instance. If you think you will want to run in a cluster at some point, you should start with a cluster from the beginning, even if it is a cluster with only one node in it.

Deploy Adequate Hardware

It is important to ensure that you deploy your application into an environment that has adequate hardware resources and an appropriate physical architecture to support your performance and availability requirements. A common cause of performance problems is trying to deploy into an environment that does not have sufficient resources.

In practical terms, you should separate components into separate tiers. Web servers/proxies/etc., application servers (i.e. BPM engine) and database should be installed in different tiers, since they require different tuning.

If you have availability requirements, you will most likely need to deploy into a clustered environment. Remember that availability is limited by the least available component in the environment. You need to ensure that all components are configured appropriately to support your requirements. For example, there is little point having a cluster of application servers sharing a single database server. While such a configuration may increase the amount of work you can process, it will not improve availability, as there is still a single point of failure in the single database server.

If you want to configure for high availability, you need to make sure every component in the environment is highly available. This means you will need redundancy, clusters and you will need to configure load balancers and virtual IP between each tier.

You should size your production hardware based on load testing of your application. It should not be sized anecdotally, i.e. it should not be based on available benchmarks or performance data of other BPM applications, since performance characteristics of BPM applications are almost always different.

Application Design Considerations

Often performance problems can be inadvertently created when the architects who design our applications and the developers who implement them are not aware of the performance implications of some of the design decisions they make. This section covers the important area of application design. This area can often contribute significant performance improvements – orders of magnitude improvements, not small incremental improvements.

Instance Size

The amount of data stored in variables when an instance of a process is running is called the 'instance size.' It is important to keep this as small as practical. A large instance size will reduce performance across the board. This is due to issues like memory usage and marshaling/parsing of the content of the instance variables. Ideally, your instance size should be less than 1 MB.

Three key strategies for minimizing the instance size are:

- minimize the number and size of instance variables in your processes,
- use separated variables (see next section), and
- store instance data in a database rather than variables (if appropriate).

Different Types of Variables

All variables defined in a process are serialized in a BLOB column in the BPM engine database. This column is updated every time any instance variable is updated.

If a variable is defined an 'external' or 'business' variable, it will be stored instead in an additional column in the PPROCINSTANCE table in the BPM engine database. 'Separated' variables are stored in a separate table (hence the name). When variables are defined as 'separated,' 'external' or 'business' variables, the database will be updated only when those variables' values have been updated or modified as part of the execution of the instance.

The aim of using these variable types is to minimize the amount of time it will take for the database transactions to commit. Remember that by default the BPM engine will execute a database transaction for every activity in every instance of every process it is executing. The exceptions to this default are when multiple activities are made part of one activity group or when greedy execution mode is enabled.

The use of local variables is also encouraged, as they will not need to be stored outside of the method where they are used.

The 'end' activity is executed as an automatic activity. This means that it will also write all of the variables to the database. If you have a large instance size, you may want to consider setting your variables to null before you reach the end activity to minimize the amount of work that will need to be done to process the end activity.

Use of Sub-processes

Creating a sub-process is a relatively expensive (time and resource consuming) operation for the BPM engine. As such, use of sub-processes should be minimized, except in circumstances where there is a good reason to use a sub-process.

Examples of good reasons to use a sub-process are to limit the scope of exception handling, create a transaction boundary or to allow for reuse of common processing logic. Creating a sub-process simply to improve the visual appearance of a process model is probably going to have a negative impact on performance.

If you do use sub-processes, you should carefully monitor the impact during testing. If the load created by unnecessary sub-processes is significant, you may wish to refactor your models to remove the sub-processes.

Handling Large Volumes

When handling large data volumes there are some important considerations to take into account when designing your application. A design that works well for a small payload may not work well for a larger payload, especially one that contains a large number of 'records' that need to be processed individually.

Batching

One key strategy for handling large payloads which contain many records is to use batching. Batching allows you to control the number of records that are processed in a single database transaction. To implement batching, your application needs to be designed such that it reads *n* number of records from database/file and processes them together. *n* should be a business parameter, so that it can be easily changed.

Transaction Grouping

When the BPM engine executes your processes, it will use a number of transactions for each process, possibly as many as one for each activity. If you know that a number of activities can be considered atomic, then you can tell BPM to group them together in an activity group and execute them in a single transaction. This can reduce the workload on the engine when there are high volumes of instances and/or large payloads.

Reuse ProcessService in PAPI client

The ProcessService is the entry point to establish sessions with a BPM Engine in a PAPI client. In the hybrid directory configuration the ProcessService stores caches for participants, group definitions, and other objects. An application that establishes connections to the Engine should reuse the ProcessService, since doing so will avoid having to regenerate all these caches. This saves a considerable amount of time (login time can be reduced from 30s to a few milliseconds, for example) by reducing the number of calls to the directory.

One way of reusing the ProcessService object is creating a static variable and storing the object in that static variable (a Singleton pattern).

Tuning the BPM Engine

The BPM Engine itself can be tuned in various ways to improve performance. The settings that you may wish to use during development of your application may be quite different to those you wish to use when you are ready to deploy your application into a production environment. This section describes the major tuning options for the BPM Engine.

Log Configuration

You should set the Engine log to the 'warning' level (or above) to minimize log-related I/O. Excessive logging can reduce the performance of the engine.

If you need to log a significant amount of data as part of the activity executions, try to set the log file size large enough that the BPM engine will not need to switch between log files often. Very large log files can severely impact performance and should be avoided. It is generally better to have more small files than less large files.

The recommended configuration is as follows:

RECOMMENDED LOGGING SETTINGS

PROPERTY	RECOMMENDED SETTING
Messages Logged from Server	Warning
Messages Logged from BP-Methods	Warning
Message Sent by Email	None
Maximum Size of Log File	10000 KB
Maximum Number of Log Files	10

If you are using the logMessage API in your process, make sure that you always use the severity argument. Do not use trace in production and avoid writing to the log in every transaction as this can cause serious performance degradation.

Audit Events Recommendations

Store events selectively on an as-needed basis and not for all activities of all processes.

Data Source Configuration

BPM data sources in WebLogic should be configured/tuned as discussed below.

BPM Engine Data Source

The BPM engine persists process state and instance variables to the database at every step in a business process. To ensure good performance, it is important to ensure that threads executing interactive or automated activities (EJBs for interactive activities and MDBs for automated activities) have enough database connections available to perform the necessary transactions immediately and do not need to wait for a connection to become free in the pool.

The data source connection pool size should be more than the maximum number of threads that can be running in parallel in the WebLogic Server running BPM.

BPM Directory Data Source

The configuration of this data source will require special attention when directory service (using FDI Components) is invoked from automatic activities or when a large number of login requests to the Workspace are expected. In practice, it has been observed that this data source often performs well with five to ten connections in the connection pool.

BPM 'To Do' JMS Queue Configuration

The BPM 'To Do' JMS queue is used by the BPM dispatcher to dispatch work to BPM engine threads for automatic activities.

It is very important that the queue is configured to be NONE PERSISTENT. This will improve performance and will avoid unnecessary redundancy in the work that needs to be processed. The Oracle BPM Engine database has a table named PTODOITEMS that host in the form of a queue (rows with timestamps) what is the work that needs to be processes automatically. It is the BPM Engine's responsibility to dispatch the work in the PTODOITEMS table into the queue at the right time, and to execute recovery tasks in the event that a node in the cluster fails.

Enable "call by reference", if needed

You should consider enabling the "call by reference" property for local EJBs. If your environment contains the Workspace application in the same instance that the BPM Engine EJBs and Process EJBs, then enabling this property will improve performance by avoiding unnecessary copying of parameter values and passing a reference instead (since they are in the same JVM).

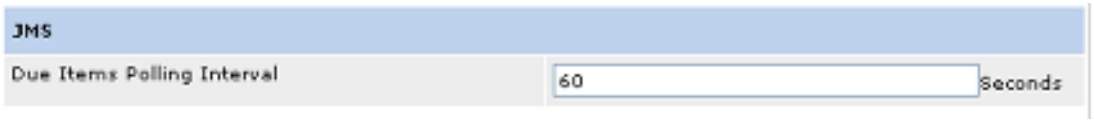
In order to achieve this, you must edit the deployment descriptors in the EAR file and add the following to all the EJBs:

```
<enable-call-by-reference>True</enable-call-by-reference>
```

JMS – Due Items Polling Interval Recommendations

This interval, measured in seconds, is used by the Scheduler to poll PTODOITEMS table. In the event that there are scheduled items with a due time, scheduler will dispatch them to the BPM ToDo JMS Queue so that they are processed by the BPM Engine MDBs.

This scheduler should be run as frequently as dictated by your business requirements, but it is recommended that this value should not be set to less than 60 seconds. Generally the higher this setting, the better from a performance point of view.



Tuning the Directory Service

BPM maintains a directory in its database. In addition, it also supports connection to a corporate directory such as an LDAP server or Active Directory. When you use an external directory as well as the BPM database, this is called the 'hybrid directory' configuration.

You should use a hybrid directory in preference to not doing so. When you use a hybrid directory, BPM caches information from the directory in its database and refreshes that information periodically or on demand. This can dramatically improve performance and reduce the load on your corporate directory. However, care needs to be taken to ensure that the refresh interval is appropriate.

http://download.oracle.com/docs/cd/E13154_01/bpm/docs65/admin_guide/modules/dir_service/t_Config_Hybrid_Dir_Service.html has information about how to configure hybrid directory service.

Master Group

In many cases, the corporate directory will contain many more users and groups than are needed by the BPM application. In these cases, you do not need to replicate the entire corporate directory.

You should use the master group functionality in preference to not doing so. Master group allows you to replicate only a part of the corporate directory into the BPM database – it controls, or restricts, which users, groups, etc. are replicated by BPM.

LDAP Queries

The configuration file for the directory (named directory.xml) contains LDAP queries that tell it where to look for users and groups and filters to apply. In some cases, these may be too general and may create additional unnecessary load. You should review these queries and tune them to be as restrictive as possible, thereby minimizing or eliminating any unnecessary work.

Ideally, you should have your users and groups stored in different DN trees in the directory. This will reduce the result set sizes of the queries. You probably only have the ability to change this in a new environment. If you have an existing directory you will most likely be unable to change this.

You should look at the filters and see if you can make them more restrictive. Consider that there may be multiple objectclass definitions in the directory. You may need to filter by more than one of these, or by an attribute on one.

Tracing

When analyzing performance problems with corporate directories, you should enable tracing so that we can see what queries are performed against the directory and how long they are taking. This information will help identify which queries are the slowest. You should work with your directory administrator to optimize these queries by adjusting your queries and/or tuning the directory server.

To enable tracing, set the system property "fuego.fdi.hybrid.ldap.debug" to "true" in the appropriate BPM component, e.g. the BPM Engine or Workspace.

Tuning the PAPI Instance Cache

PAPI APIs are used by BPM clients and the out of box Workspace application to communicate with the BPM Engine. BPM provides a client side cache called the PAPI cache to improve performance of these APIs. There is one cache per JVM. The cache can be either in the OPEN or CLOSED state. If the cache contains all in-flight instances, then it is in CLOSED state (by definition). A closed cache can improve performance very significantly (compared to an open cache).

The workspace instance cache should be tuned as part of load testing. The optimal condition for the cache is "closed." It will be "open" when the server starts up. The aim of tuning is to ensure that it reaches the closed state as quickly as possible. It is possible to "force" the instance cache into the closed state quickly by using a script/program to access all in-flight instances using the PAPI interface, or by defining a special user who has all available roles, and accessing the workspace using this user, thereby forcing the system to read all instances into the cache. When the cache is in the closed state, there will be a message in the Workspace log stating that the "Cache is now closed." Once the cache is in the CLOSED state, it is automatically synchronized with the BPM Engine and is updated with new in-flight instances that are created.

You must tune the instance cache size parameter (fuego.workspace.papi.instancesCacheSize) in the workspace.properties configuration file so that the cache is large enough to hold all in-flight instances in the system (i.e. tasks that are not completed). For PAPI programs, this property should be specified when creating an instance of the ProcessService object. You must also ensure that the JVM Heap size is large enough to hold the instance cache. You should multiply the size of an instance by the number of instances to calculate the cache size and the amount of Heap required.

Important point about PAPI cache

The PAPI cache stores information about in-flight instances only, so if you try to get info of completed/aborted processes, then cache will not be used and call will go to BPM Engine.

Participant cache

The BPM Engine also keeps a (separate) cache of participants, so that it does not have to go to directory each time it need participant information. This cache is populated as user information is first accessed or as users login. There is a 'participant cache' parameter on the 'execution' tab in Process Administrator that controls the cache refresh period. You should set this carefully so you have the optimum balance between data being up to date, and refreshes not occurring too frequently. This optimal setting will be different in each environment.

Keep in mind that some APIs go directly to the directory and bypass this cache.

Cache ProcesService object

ProcessService is first API call made by PAPI client programs and it caches participants, group definitions, etc., and establishes a connection with the BPM Engine. So reusing ProcessService avoids the overhead of regenerating the cache and reuses the connection. Use a singleton to cache and reuse the ProcessService object.

Testing

Testing is a very important and often overlooked part of performance tuning. There is a significant amount of risk involved in transitioning an application into production without doing adequate testing. Many performance problems will not show up in development and only become apparent when you move to a production-like environment.

You should conduct testing of all major components of the system before production cutover. You should test that the application functions correctly in the production environment, that it functions correctly in a cluster (if you are using one) and that it can handle sustained load.

In addition to testing, you should use the best practices advice in this whitepaper to configure your production environment so that performance or load related problems are less likely to occur.

If your project timelines are such that you cannot perform adequate testing before production cutover, you should plan to perform additional, detailed testing as soon as possible (e.g. starting immediately after production cutover) and quickly implement fixes to any issues discovered.

Testing may be complicated by unavailability of test instances for some back end systems or limited (or no) access to test hardware. You should identify these risks to your project manager and/or sponsor as early as possible so that they can be understood and mitigated.

Functional Testing

You should conduct end-to-end functional testing on the either the actual production environment, or an environment that is as similar as possible to the production environment, as early as possible before production cutover. This is especially important if you are developing on a different platform than you will be deploying your application to in production.

All processes, services and dependent artifacts should be deployed on either the actual production hardware, or hardware that is extremely similar. They should be deployed with the same operating system version and patches that will be used in production. The versions and configuration of all Oracle software should be the same as production, and the same logical architecture, e.g. Load Balancers, etc., should be used in the test environment.

Each process and service should be tested, from end to end, to ensure that all components of the system will function as designed in the production environment.

Functional testing aims to find any components that will not work in production. For example, BPM allows you to use COM objects (from a PBL script). This functionality is often used to create a Microsoft Word document in a process. It is not possible to use COM on a non-Windows operating system. This means that an alternative approach would be needed. For example, you may choose to use the COM bridge on a Windows server or replace the COM object with a Java library with equivalent functionality.

Functional testing would identify all issues of this nature, so that they can be fixed in advance of production cutover. Failure to conduct thorough functional testing could result in these kinds of issues being discovered when production cutover is attempted, leaving no time to design, implement and test a fix.

In addition to testing the runtime components of the system, you should also test the deployment process to ensure that it will work. Specifically, handling of environment-specific customizations (e.g., endpoint addresses) should be tested and understood. The aim here is to reduce the risk that deployment to the production environment may not work as expected.

Cluster Testing

A second key area to test before production cutover is to ensure that the system functions correctly in a clustered environment. Development and unit testing are usually done in non-clustered environments. It is important to ensure that the system still functions correctly in a clustered environment, and also to ensure that the clustered environment is actually being utilized and providing additional scalability to the system.

Cluster testing should start by validating that the system can be deployed into a cluster and that the deployment process is well understood and repeatable. Next, you should validate that the system functions as expected in a cluster; that is, make sure you can still run all of the processes and services in the cluster and that they function the same as they did in a non-clustered environment. Finally, validate that the cluster is actually being used – check that requests are in fact being sent to every node in the cluster and not all to the same node. If you have configured, or expect session affinity, check that your user sessions are in fact returning to the same node with each subsequent request in a conversation. Check that the endpoint address for service calls, and the callback addresses for responses from asynchronous services are correctly configured to point to the load balancer addresses, not individual nodes in the cluster.

Since part of the benefits of deploying in a cluster is resiliency, you should do additional testing to validate that the cluster continues to operate correctly after a node failure, and that it recovers correctly when that node is restarted.

Load Testing

Discuss issues with doing a valid and meaningful load test.

The third key area to test before production cutover is load. Load testing requires a significant amount of effort to conduct correctly, so that the test results are meaningful and can be used to correctly size the production environment.

Load testing can be difficult and time consuming. It may be necessary to create "stubs" or small simulators for back end systems, it may be necessary to create a significant amount of test data, and it may even be necessary in some cases for the stubs to have some persistence and simulated business

logic in them, so that they can drive the process down the paths that need to be tested. It may also be necessary to write utilities to create load by invoking the process and processing human tasks to keep the process running. When using real instances of back end systems, it may be necessary to set up special "test" accounts and this task can be nontrivial.

Load testing should ideally be conducted using test instances of back end systems, in order to get more accurate measurement on latency, throughput and capacity. It is not uncommon to find issues where backend systems are not properly handling the load from the middle tier applications. Load testing against the actual systems is the only way to uncover these issues before production. If this is not possible, then load testing should be done with stubbed-out back ends; that is, small simulators replacing the real back end systems. These should behave similar to the real back end systems, as much as possible.

It is also important to ensure that the BPM Disposer task is running during load testing in order to ensure that its overhead is incorporated into any measurements.

Load testing should be conducted for both manual (user entered load) as well as bulk loads. It is important to make sure that the load generated is representative of what is expected in production. It is also important to conduct tests of both peak and sustained load. You should run a moderate load for an extended period, e.g. three days. This should assist in detection of any memory leak issues. Load should be generated using a tool like HP LoadRunner, Apache JMeter or similar so that tests are reasonably well-defined and repeatable.

Each of the following recommendations should be observed during load testing. It is important to note that most of these will require some specialist skills to collect and interpret the results. Oracle recommends that you identify one or more individuals in your team who can learn these skills, as they will be required each time you bring on more users or release another phase into production. These skills will be needed in order to ensure that the new workload will not overload the production servers, and to help plan for capacity growth and to tune the environment based on the workload.

- The BPM audit trail and activity level metrics should be enabled during load testing, and the output used to help identify bottlenecks in the process flow.
- JRockit Mission Control should be used to help identify issues at the JVM level, e.g. garbage collection efficiency, memory leaks, thread contention, etc. It is highly recommended that you conduct a JRA recording during the load testing.
- Oracle Database Automatic Workload Repository (AWR) reports should be run and the output used to help identify any performance issues at the database level. You should look for queries that are not using indexes (i.e. are performing a table scan) and ensure that any necessary indexes are created. Missing indexes can cause significant performance overheads. Standard database tuning should also be performed, that is tuning of the SGA size, connections and file I/O overhead. You should configure the database to store BLOBS in separate tablespaces.

JVM Tuning

BPM is supported on various JVMs. The choice of JVM can have an impact on performance. The general rule of thumb for BPM is to use Oracle JRockit in preference to other JVMs. You should use the latest stable release of the JVM that is available. You should use the same JVM and the same version of that JVM in each of your environments.

The second main consideration, if you are running on a 64-bit platform, is whether to use a 32-bit or 64-bit JVM. The general recommendation is to use a 64-bit JVM unless there is some good reason not to. 64-bit JVMs do consume more memory than 32-bit JVMs, in part because they use twice as much memory to store each reference. If this is an issue, you should consider using the UseCompressedOOPS (or equivalent) JVM setting, which will cause the 64-bit JVM to use less memory for references. JRockit does this automatically, other JVMs will require a parameter.

As a general rule of thumb, you should use a 64-bit processor architecture and operating system in preference to a 32-bit environment.

Some applications can benefit from having a larger number of smaller JVMs. If your application fits into that category, and it does not need to keep as much data in memory in any one JVM, you may consider using a number of 32-bit JVMs.

Because of the large amount of variation in applications built using BPM, it is not possible to give a recommendation that will work for every case. It is important that you test your application and tune your JVM. This should be an iterative process to find the best configuration for your own environment and workload.

When tuning your JVM(s) you should consider sizing of the heap and permanent generation (in HotSpot) as well as the garbage collection algorithm selection and configuration.

Recommended Settings

There are some basic JVM settings that are recommended in most cases. You should consider using these settings in all of your environments. The table below summarizes the recommended settings for HotSpot and JRockit, which are discussed in the following section. If you are using another JVM you should look up the equivalent settings for that JVM. You should only use a JVM that is supported by BPM on the platform you are using.

RECOMMENDED JVM SETTINGS

JVM	SETTING	PURPOSE
Both	-XX:PrintGCTimeStamps -XX:PrintGCDetails -Xloggc:filename.log	Enable garbage collection logging. Logs will be written in to the file named in the –Xloggc setting.
HotSpot	-XX:HeapDumpOnOutOfMemoryError	Request that the JVM dump the heap if an OutOfMemoryException

RECOMMENDED JVM SETTINGS

JVM	SETTING	PURPOSE
		occurs.
JRockit	-Djrockit.oomdiagnostics=true -Djrockit.oomdiagnostics,filename=x	Print diagnostic information about the heap to stdout and to the named file if an OutOfMemoryException occurs.
HotSpot	-server	Tell the JVM to run in 'server' mode.
Both	-Xms -Xmx	Set the initial and maximum heap size. Generally we recommend setting them the same.
HotSpot	-XX:PermSize -XX:MaxPermSize	Set the initial and maximum size of the permanent generation. Generally we recommend setting them the same.
HotSpot	-XX:+UseCompressedOOPS	(Optional) Tell a 64-bit JVM to use less memory for references.
JRockit	-XXcompressedRefs	(Optional) Tell a 64-bit JVM to use less memory for references.

Garbage collection logging is very useful when you need to tune your JVM. The overhead of collecting this information is very low. It is recommended that you turn on garbage collection logging in all environments including production. If you do have a need to do some performance tuning, it is helpful to have to have actual garbage collection data from your production environment. Leaving garbage collection on all the time will mean that you will always have access to data if you need it. If a problem occurs, you will not need to attempt to reproduce it in order to capture garbage collection data.

Similarly, if you suffer an outage due to insufficient memory, it is extremely helpful to know what was happening when memory was exhausted. You should use the settings in the table for your JVM to collect and save information to help you investigate the cause of memory exhaustion outages.

If you are using the HotSpot JVM you should explicitly tell it to run in 'server' mode on 64-bit machines. This setting will enable various optimizations that will improve performance on modern 64-bit multicore processor architectures.

In large JVMs (those with more than 2GB heap is a good rule of thumb) working out if the heap needs to be resized and resizing it when necessary can consume a lot of resources. We generally recommend that the initial and maximum heap size be set the same to avoid this additional overhead.

Similarly, in the HotSpot JVM we recommend setting the initial and maximum size of the permanent generation the same for much the same reason.

If you are running in a 64-bit environment and your heap is approaching the size of the available memory, you may wish to enable compressed references to reduce the amount of memory consumed by the JVM. This setting is listed as optional as it may or may not have a positive impact on

performance depending on your application. However, it will reduce the memory footprint. If this is a consideration in your environment, you should conduct some testing to analyze the impact of using this option. Note that JRockit will use compressed references by default in some circumstances.

Heap Size

As a general rule of thumb, you should try to size your heap to be at least twice the size of your instance cache. The maximum heap size plus the maximum size of the permanent generation should be less than the available physical (real) memory after the operating system and any other applications memory needs are taken into account.

WebLogic Server Considerations

The Performance and Tuning Guide (see link below) provides extensive information about best practices for configuring and tuning the WebLogic Server and JVM. We recommend that you review this document, at least Chapter 2 ("Top Performance Areas"), and implement the recommendations. In particular, you should tune the JDBC connection pools and the MDBs on the BPM "TODO" JMS queue.

Standard JVM tuning should be carried out, with a focus on heap size and garbage collection.

http://download.oracle.com/docs/cd/E14571_01/core.1111/e10108/toc.htm

Non-persistent JMS Store

For the WebLogic Server instances that are running BPM, you should make sure that the JMS store is non-persistent. BPM only uses the "ToDo" queue as a dispatching mechanism and it will persist the data anyway, so having a persistent JMS store will add additional unnecessary overhead.

Transaction Logs

Consider storing WebLogic Server transaction logs on shared disk. This will help to improve reliability and fault tolerance.

Database Tuning

BPM uses a database to store instance state and also to store its directory. The performance of the database can have a significant impact on the performance of BPM, so it is important that the database is also tuned. If the nature of the workload on the BPM server changes significantly, you should check the database performance and retune it if necessary.

Automatic Workload Repository

The Oracle database includes a feature called the Automatic Workload Repository which will automate collection of statistics about database performance. This information can be used for database tuning. In order to run the reports you will need to license the Database Tuning option.

When you have a representative workload running on the BPM server, ideally during load testing, prior to production, you should run AWR reports for a suitable period, e.g. three to seven days. The AWR reports will highlight any poorly performing queries. You may need the assistance of your Database Administrator to interpret the information in the report and make the necessary changes to address the issues identified.

Missing Indexes

One issue that is often observed and easy to fix is missing indexes. If you find that you have queries running that are performing table scans (i.e. reading a whole table) because there is no index available, then you should create a suitable index that would allow the query to be processed without a table scan. This can create a significant performance boost.

Use Separate Tablespaces for BLOBs

If you are storing large BLOBs in the database, you should consider creating a separate tablepsace for the BLOBs. If you have different types of storage available, you may want to consider where to place this tablespace on your storage farm in order to get optimum performance of queries. You should discuss this with your Database Administrator.

Conclusion

We have seen that there are many options available for tuning Oracle BPM 10g and have provided some guidelines and recommendations about how to use the options to improve performance of your system/application.

It is important to understand and remember that performance tuning is an iterative exercise. It is important that you get in to the habit of measuring, tuning and repeating.

It is also critically important to understand the role that your infrastructure and application design decisions can have a major impact on the performance you see, and to ensure that these areas are not neglected when attempting to tune the system.

In general, 'tuning' of the system would be expected to contribute performance improvements in the range of up to 20%. The large, orders of magnitude performance improvement usually come from making changes to the application design and infrastructure.

Oracle is committed to developing practices and products that help protect the environment.

Performance Tuning for
Oracle Business Process Management
Suite 10g
July 2011
Authors: Sushil Shukla, Mark Nelson

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com

SOFTWARE. HARDWARE. COMPLETE.