ORACLE

FUSION MIDDLEWARE

BUSINESS PROCESS MANAGEMENT

An Oracle White Paper
March 2013

Performance Tuning for Oracle Business Process Management Suite 11g

ORACLE

This white paper is current up to version 11.1.1.6 which was released in February 15, 2012. It will be updated to reflect new best practices and any different recommendations for newer releases as they are identified and confirmed in production environments.

The recommendations in this paper are also appropriate for 11.1.1.7, however new features of 11.1.1.7 are not covered in this paper as there has not been enough real world usage of those features at the time of writing to observe best practices.

Introduction

Many organizations have built and deployed mission critical systems using Oracle Business Process Management Suite 11g. 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 is not intended to be a substitute for the product documentation. This white paper should be read and its recommendations considered in conjunction with the *Oracle Fusion Middleware Performance and Tuning Guide* that is included in the documentation set for BPM.

This white paper presents a set of tried and tested performance tuning 'best practices' – collected from observations of successful BPM deployments at many different organizations over an extended period of time. This white paper was collated from the real world experiences of the people who tune some of the largest and most critical BPM implementations in the world, and reviewed by the people who develop BPM and the people who run Oracle's benchmarks and performance tests on BPM.

While every application is different and has its own set of performance tuning challenges, this paper attempts to present a set of guidelines and a common sense approach that will hopefully 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 (i.e. maximum throughput), 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.

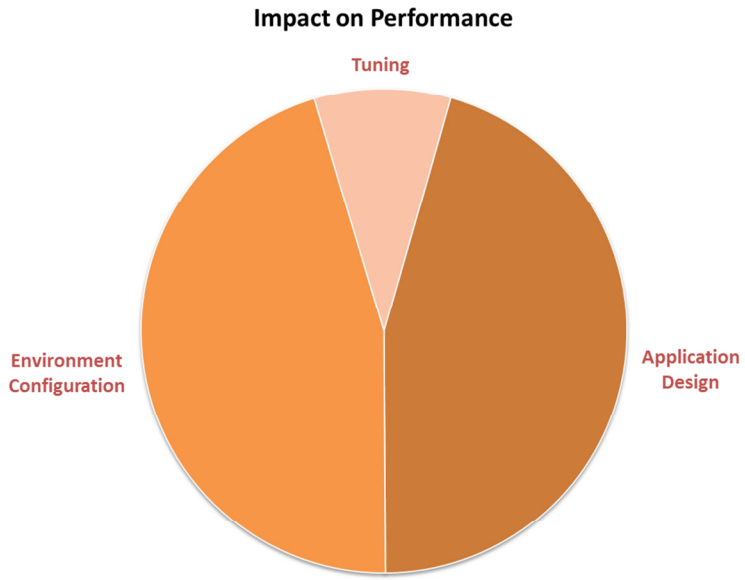
Understanding the Goal

In order to maximize performance, it is necessary to monitor, analyze and tune all of the components that make up your application. This whitepaper describes the 'knobs' that can be adjusted and when and how it might be desirable to adjust them.

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

- Configuration of the BPM environment, including the deployment architecture,
- 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, and
- Tuning of disk and network I/O.

The first two items – correct configuration of the environment and good application design – typically have the largest impact on performance. These are the areas where large, orders of magnitude improvements in performance are typically achieved, and as such they take up a large part of this document. The others areas are also important, but these should be considered 'fine tuning' as they tend to produce relatively small improvements of up to around 20%. For this reason, it is very important that environment configuration and application design are not neglected.



It is also important to understand that performance tuning is an iterative process. Tuning should consist of making a small adjustment, measuring the impact, and then performing analysis and making

another adjustment, and so on. Due to the vast differences in applications that customers build using Oracle Business Process Management 11g, there are no global solutions that work well in every environment. Improving performance is a process of learning and testing.

Typical Issues

Experience shows that there are some common areas where bottlenecks tend to occur. Being aware of this can help us to focus initial tuning efforts in the areas where they are most likely to deliver greatest value. The bottlenecks that have been observed tend to vary based on whether the workload is mostly human oriented or mostly system oriented.

When the workload is mostly processes that are automated (system oriented), the commonly observed issues are:

- Database latency, especially when there is a high rate of instance creation, and especially the first time the system is placed under load. This is observed both on the tables and sequences. The audit tables in particular are a hot spot,
- Exhaustion of the invoke and engine thread pools, especially when there are a lot of invokes and callbacks, and
- Exhaustion of memory.

When the workload is mostly process that are driven by human interaction, the commonly observed issues are:

- Lack of capacity (CPU, memory) on the web tier, i.e. the managed server(s) where the ADF human task user interface applications are running,
- Slow BPM WorkSpace login due to failure to tune LDAP, and
- Slow BPM WorkSpace performance due to failure to create indexes in the database for Flex Fields.

Performance Tuning Approach

When ready to start performance tuning, it is advisable to adopt a documented, repeatable process which ensures that the state of the environment is always known, and there is clear evidence of what improvements have been made, and what changes brought about those improvements. This generally allows reproduction of tuning outcomes and performance gains in other environments.

If an organization does not have a defined process, the process illustrated in Figure 1 as a starting point which can be adapted to the environment as needed.

Before starting, the success criteria should be defined, so that it will be possible to know when the exercise is finished. That is "good enough" should be well understood by all stakeholders *before* starting

performance tuning. These criteria may be expressed in terms like the examples below, and they should be specific and measurable:

- Number of process instances that can be completed in a given period of time,
- Number of process instances that can be open simultaneously,
- Number of users that can concurrently be using the workspace to process tasks – this kind of metric should include 'think time' to produce a hit rate that simulates actual users,
- Maximum amount of time to complete a given number of instances, or
- Maximum amount of time needed to recover after a node failure.

The end state for a performance tuning exercise is acceptable performance, together with demonstrated stability. Avoid tuning the environment so finely that an unexpected workload spike can cause the JVM to get behind and be unable to recover.

Important considerations when running a performance test

It is very important that individual performance test runs are as similar to each other as possible, to ensure that it is actually meaningful to compare and draw conclusions from the results.

For each performance test run, ensure that the environment is warmed up and has reached its steady state before starting the actual performance test. Measurements must be taken during the steady state.

Performance can differ markedly between cold and warm environments because of issues like the impact of optimizations that are done by the JIT compiler after a number of repetitions of a particular piece of code, the optimizations done by the database after a number of queries have been executed, the settling of objects in the various parts of the JVM heap over time, and so on.

Ideally, a baseline should be established at the start of the performance tuning exercise. In each iteration (test run) change only one parameter, and compare the results against the baseline and previous test runs.

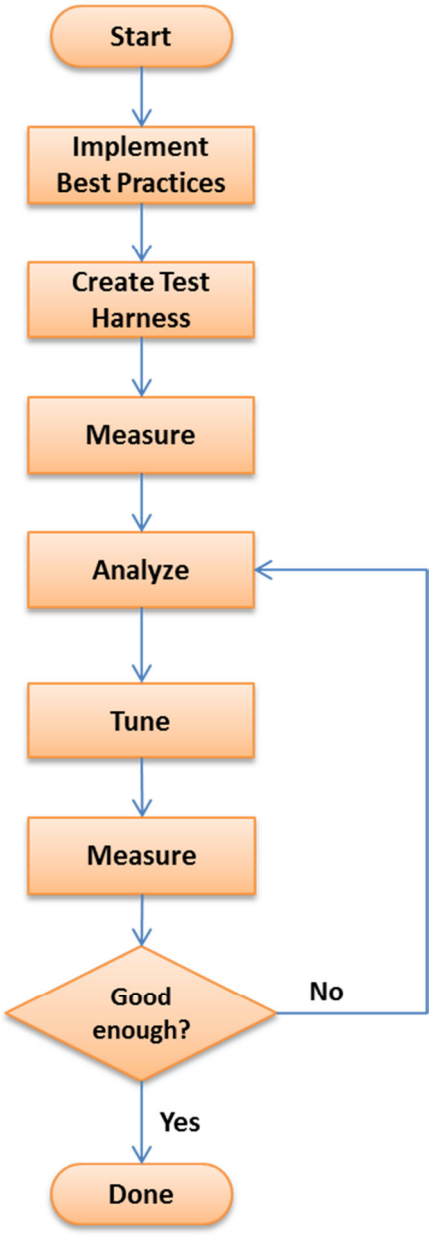


Figure 1. Performance Tuning Approach

Implement Best Practices

Adopt and implement the best practices advice in this document. There is no point in repeating the same learning. Start with your environment in a sane state.

Create Test Harness

Create a representative workload and a repeatable test harness that allows execution of that workload over and over again, so that the impact of changes made can be measured against the exact same set of inputs.

Measure

This one may seem obvious, but it is necessary to take measurements so that a point of comparison exists. Measurements should include the metric(s) of interest (e.g. throughput, response time), as well as statistics from the environment in its steady state (e.g. CPU, IO, JVM statistics.) It is important that a starting measurement is taken to establish a baseline before starting a tuning exercise.

Analyze

Examine the last test run. If it was clean (there were no errors that would make the collected data meaningless) then analyze the data and build a new hypothesis to test.

Tune

Tune the system. Make sure changes are documented so that the environment will always be in a known state, and only change one thing at a time so that it will be clear what caused the measured differences. Note that 'tuning' may include making application or environment changes.

A good approach to iterative tuning is to consider the environment as a 'network of queues,' as illustrated in Figure 2 which presents a simplified view of a typical production environment, i.e. not every component of the environment is shown on this diagram.

The method employed in the 'network of queues' approach is to look at the environment as a network of interconnected and interdependent workers, and to recognize that queues can form before any

worker that is not able to keep up with the amount of work being presented to it. Workers may be unable to keep up because they have insufficient resources (e.g. CPU, memory, etc.) or because they have more constraints (e.g. degree of parallelism) than other workers.

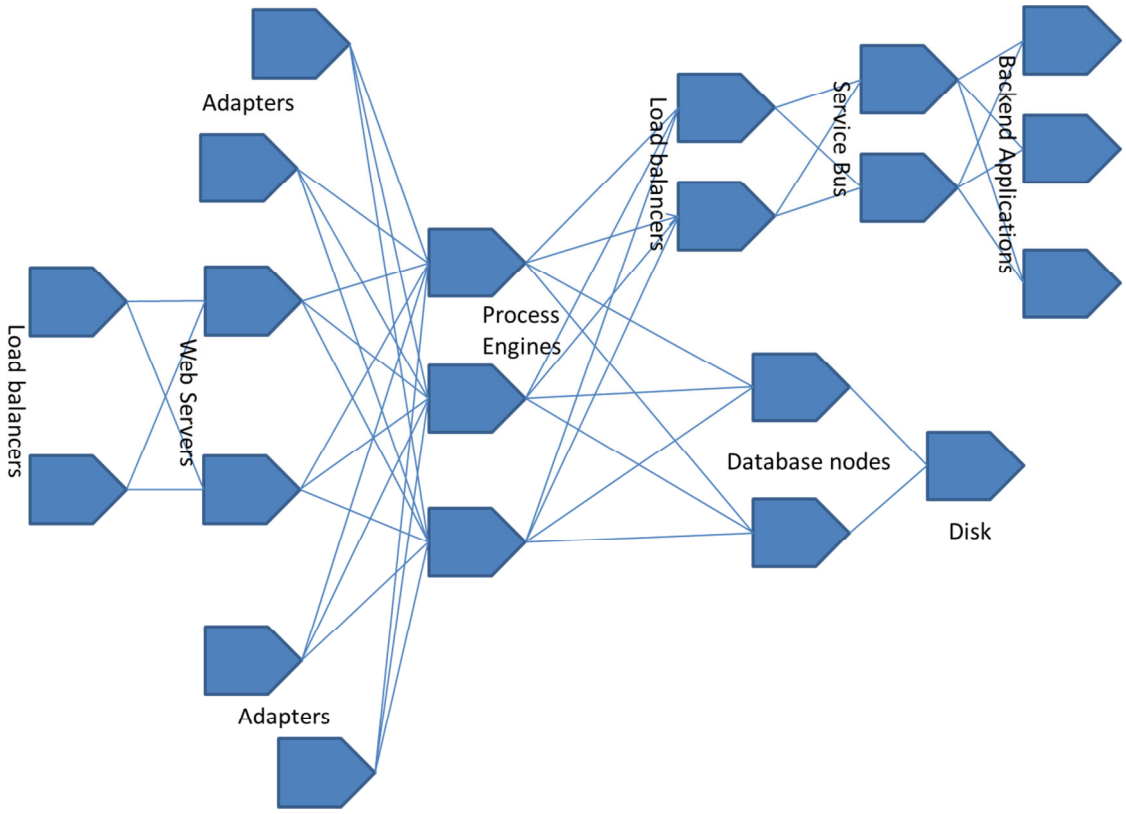


Figure 2. A 'network of queues'

This diagram is not attempting to represent every possible queue in any given environment; rather, it is intended to be a starting point to work from. Understood what the network in the particular environment of interest looks like, and become familiar with how to measure and tune each queue.

It is necessary to run a defined workload (or the real production workload if tuning a production environment) through the environment with appropriate monitoring so that it is possible to determine when and where the queues form. Wait for resource exhaustion (CPU, disk, memory, bandwidth, threads, heap, etc.) and then note where the worst queues occurred, and perform tuning to alleviate the problems that caused those queues.

For example, if it is observed that the BPMN Service Engine is not able to keep up with the volume of work coming in from the web servers and adapters, monitoring may show that there are a large volume of messages coming in from a JMS adapter. To alleviate this problem, you might add more threads to the BPMN Service Engine, or alternatively reduce the number of threads in the JMS adapter. Either of these actions should either fix the problem, or move it (to the right or left of the diagram, respectively).

After making changes – and they should be small, incremental changes – test again and measure the impact of those changes. Make as few changes as possible in each iteration, ideally only one. Large changes are likely to cause unpredictable results that are difficult to interpret.

Continue iterating until all of the queuing has been driven out of the network, until performance is deemed acceptable, or until performance cannot be improved any more.

The key benefit of this approach is that it forces the practitioner to remember that the performance of any one component is in fact dependent on the performance of many other components and helps them to understand those relationships. Alleviating stress on one component may cause stress on another component, which was hidden by the original problem, to become apparent.

Automation is essential here – it is important to be able to run the same workload over and over again, so that information collected is relevant and to avoid chasing a moving target. Failure to run the same workload will make attempts at tuning BPM very difficult.

While this may seem obvious, the importance of adopting a holistic perspective and approach to performance tuning cannot be overstated.

Consider an example: A financial institution, through its online banking portal, allows customers to apply for loans. The loan products are customized based on customer's banking history, credit rating and various other factors. The product customization and the back end processing, once a loan application is received, are implemented using BPM. The paramount performance requirement for this use case is to minimize latency of screen rendering to the customer. Consider that the decision on what screen to render next may be made by BPM, which in turn needs to invoke various back-end services in order to make that decision.

In this scenario, focusing tuning efforts on BPMN Service Engine optimization alone would not be the optimal or desirable approach, since at the very least the following areas should also be considered:

- The impact of firewalls and network hops,
- Network optimization across the application and web tiers,
- The impact of SSL usage,
- Overhead/latency added by every component – load balancer, web server, etc., and
- Response times of the back end services.

This example demonstrates the need to think of performance holistically.

Environmental Considerations

Many performance issues can be avoided by ensuring that the environment is well suited to the intended task. Prevention is almost always better than the cure!

Use Current Versions

Environments should be built using current versions of software unless there is a good reason not to do so. Using the current software versions ensures avoidance of any issues that are already addressed by patches that Oracle has released for issues encountered by other customers.

Practically speaking, this means environments should be built with the latest available production release in the 11g release stream, plus the latest available recommended patch set (if any).

Patches should be deployed into a non-production environment and tested before rolling patches into production. Ideally, this non-production environment should be a staging environment, not an environment that is used for development or other testing purposes. The environment should be as close as possible to an exact replica of the production environment.

Application Server

The recommendations in this whitepaper assume that BPM is being run on WebLogic Server. Some recommendations may not apply if a different application server is used.

Deploy Adequate Hardware

It is important to ensure that the environment in which applications are deployed has adequate hardware resources and an appropriate physical architecture to support the 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, separate architectural components should be separated into separate tiers – the web servers, application servers (i.e. the BPM servers) and database should be installed in different tiers, since they require different tuning. If the WebCenter portal and content management components included in BPM 11g are being used, they should also be installed on different machines.

Clustering

When the application has availability or reliability requirements, consider deploying into a cluster in preference to using a single server instance. If it is believed that there might be a need for a clustered

configuration eventually, consider deploying into a cluster from day one, even if it is a cluster with only one node in it. This will make it easier to add more capacity later on.

Remember that availability is limited by the least available component in the environment. It is essential to ensure that all components are configured appropriately to support the requirements. From an availability standpoint 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 the environment is to be configured for high availability, it is necessary to ensure every component in the environment is highly available. This means that redundancy, clusters, load balancers and virtual IP between tiers will be required.

Production hardware sizing should ideally be determined based on the results of load testing of the actual 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 different BPM applications vary greatly.

Auditing and Logging Configuration

It is important to consider the amount of data that will be created by logging and auditing, and to plan for the relevant file systems to have enough space. In particular consider the following:

- NodeManager logs, stdout/stderr files from managed servers and nohup output files are not rolled by WebLogic Server. If there are space constraints, consider using a utility like 'logrotate' (see Linux man pages) to roll or otherwise manage these files,
- When enabling additional logging, e.g. diagnostic logs, be aware that these may create a very large amount of data very quickly, and
- Ensure that there is adequate space in the temporary file system (usually **/tmp**). A good rule of thumb starting point would be 2 GB of free space, however file system usage should be monitored to determine what amount of space is adequate for a particular environment. If there is insufficient space, the managed servers and AdminServer may not start.

Deployment Architecture

The term 'deployment architecture' in this section is taken to refer to the way each of the components has been laid out with respect to the physical architecture of the system, i.e. on which physical node each component has been placed. Having the right deployment architecture is critical; the difference between a good and bad deployment architecture can impact application performance by one or more orders of magnitude. Ease of administration and scalability can also be attained from a well thought out and implemented deployment architecture.

One standard deployment model has been discussed extensively in the Enterprise Deployment Guide included in the Oracle Fusion Middleware documentation set. Review that model and the principles behind it. Consider whether that model is a suitable starting point for designing the deployment architecture.

This section enumerates some additional considerations for a deployment architecture which is optimized specifically for performance.

ADF Applications

When defining a deployment architecture for an application that uses ADF, consider whether to deploy ADF-based applications (such as, the BPM Workspace and any ADF applications created as Task Forms for Human Tasks) on the same managed server(s) as the BPMN Service Engine, or on different managed servers. Unless there are a very small number of human tasks, place these applications on separate managed servers. If using WebCenter to run BPM Process Spaces, deploy the ADF applications on to the WebCenter managed servers (as opposed to the BPM managed servers).

A key advantage of placing ADF applications on a dedicated managed server is the flexibility to tune them independently, according to their own performance requirements. It is likely that managed servers hosting ADF-based applications will need to be tuned differently to those hosting other components.

SSL usage

Be aware of where in the environment SSL encryption is being used. SSL usage typically comes with a fairly large overhead (something around 30% extra CPU usage). Ensure that SSL is only being used where it is necessary. Note that some hardware platforms incorporate cryptographic offload processors which may reduce or remove the observed overhead.

For many deployments with good network-level security, it is usually deemed sufficient to use SSL just in the DMZ, terminating at the load balancer.

Unnecessary SSL usage will be detrimental to performance. For example, BPM Workspace access is likely to be significantly impacted by the use of LDAPS (LDAP over SSL) compared to plain LDAP communication to the Directory Service.

Policy Store

For production deployments it is highly recommended to use the database or OID for the policy store. The file-based policy store (**system-jazn-data.xml**) is not suitable for production environments. While workarounds like manual synchronization of this file across the Admin Server and managed servers, hosting the file on shared storage, etc., might be viable for some deployments, it is likely that keeping policies in this file-based store will quickly become a bottleneck area.

Moving the policy store to an Oracle database appears to offer the best performance when compared to the alternative approaches. This may or may not be true in every environment, so it is advisable to do some independent testing to verify if this is the case in any particular environment.

Additional Considerations for Development Environments

Development environments have different requirements when compared to production environments, and they should be tuned differently. For example, it is often desirable to have detailed logging in development environments but it is unlikely that this level of logging would be considered in production. Additionally, in development environments, there is often a desire to minimize the memory footprint and server startup time, as it is likely that there will be more frequent server restarts.

Admin Server-only or Managed Servers

When creating a WebLogic domain to run BPM, the 'BPM for Developers' profile is presented as an option. If this profile is selected (as opposed to the 'normal' one), the resulting domain will be configured with all of the necessary components installed on the Admin Server, and there will be no managed servers created in the domain. This configuration allows significant reduction in the memory footprint for the development server.

An Admin Server-only domain should be used in development environments unless there is a good reason not to do so. An example of a good reason not to use it is a need to do some testing on behavior of a component in a cluster. In this situation, it would be necessary to have the managed servers.

Production Mode

WebLogic Server and the SOA/BPM Infrastructure both provide the option of running in 'production' or 'development' modes. Set these to 'development mode' in development environments. These options provide additional logging but do have a negative impact on performance.

Remember to set these to 'production' when doing any performance testing or benchmarking of the application, otherwise the results will not be representative of what would be seen in a real production environment.

JDeveloper Tuning

The default memory settings for JDeveloper can be a little low for doing SOA/BPM development, especially when creating a lot of Human Task forms. Consider running JDeveloper on a 64-bit JVM and increase the defaults, in particular the Maximum Permanent Generation Size and Maximum Heap Size, by updating the following settings in your JDeveloper configuration file (**ORACLE_HOME/jdeveloper/ide/bin/ide.conf**) to appropriate values for the environment – the sum of these two values plus 512 MB should be less than the available memory.

```
AddVMOption  -XX:MaxPermSize=768m
AddVMOption  -Xmx2g
```

Note that the maximum process size for 32-bit JVMs on Windows is limited to 2 GB (or 4 GB on some other operating systems), so it would be necessary to use smaller values on such JVMs. Remember that the JVM uses some native memory and permanent generation in addition the heap. Note also that the benefit observed from these particular settings varies from operating system to operating system.

If planning to create a lot of Human Task forms or other ADF user interface components, consider changing the default editor for JSPX files so that they open in 'source' view. This will greatly reduce the time taken to open JSPX pages in JDeveloper.

To do this, select Preferences from the Tools menu, then select File Types in the navigation tree on the left hand side. Open the Default Editors tab on the right, find JSP Source and change the default editor to Source using the drop down box at the bottom.

Application Design Considerations

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

BPMN Top Design Considerations

Minimize the amount of data stored in the process instance. Obviously, there is a tradeoff between the cost of storing data locally compared to storing keys and retrieving data for use within the process, which needs to be considered.

A reasonable starting point would be to model the process state to hold only values that are needed to control the process flow and keys to get any other (external) data on an 'as needed' basis. If retrieval is too frequent/slow, or the systems holding that data are not always available, then move more data into the process.

Apply a similar principle to designing the data you send to a human workflow task activity through careful task payload design. As each property of the task data definition is stored as a separate row in the database, a complex data object should be used in preference to a large number of separate attributes.

Do not auto-initialize variables that are always set from incoming messages.

Always model data objects in a process in the smallest scope where they are needed. In cases where a large payload is received and/or returned by a service, consider creating a sub-process to limit the lifetime of the data.

Do not duplicate variables that hold analytics. Values holding analytics can be referenced as process data objects. It is not necessary to create process data objects and matching measures, counters, and/or dimensions along with scripts to copy data. Analytics are valuable, but they do carry a small overhead. Use them only when they are needed.

Whenever possible, refactor reused logic into, and then call a 'reusable process' to minimize duplication of code across processes. A reusable process has Start and End nodes with the None implementation type and they are not visible outside the composite in which they are defined. They are invoked using the Call activity. The main advantage from the performance point of view is that reusable processes are run in the same thread as the calling process, so they do not incur the penalty of creating a new process instance. Note that the calling process will wait for the called process to finish before it continues.

From a purely performance point of view, do not use loops over activities to process 'sets' (arrays, collections) of data. In cases where the complexity of the transformation is such that it is difficult to

implement in XPath, consider using a Spring component or EJB instead and implement the transformation logic in Java. If it is important that business users can read and understand the models, this consideration may overrule this recommendation. However, Java should be used at least to handle large collections if nothing else.

It is important to use utilize error events to contain errors. Do not throw errors to control normal/expected process flow. This is similar to the principle in Java and other high level languages of not using exceptions to control program logic.

Process Complexity

As a general rule of thumb, it is not desirable to have a BPEL process with more than one hundred activities in it or a BPMN process with more than about fifty activities. If more activities are needed, consider breaking the process down into smaller sub-processes and chaining them together. These are not hard and fast numbers – determine suitable numbers based on testing.

Having a large (complex) process not only makes it harder for people to understand what the process is doing, and therefore increases the likelihood of logic errors, it also increases the amount of memory that the process engine needs to describe the process state, manage variables, etc., and makes recovery more difficult and resource consuming.

Synchronous or Asynchronous

As a general rule, make processes and other artifacts asynchronous unless there is a good reason to make them synchronous. While asynchronous components do attract dehydration overhead, it is much easier for the system to load balance them across available nodes in the cluster and to handle rollback and retries when necessary.

Synchronous processes (BPEL or BPMN) should never have:

- Mid-process breakpoint activities (e.g. receive, onMessage, onAlarm, wait),
- A non-idempotent (i.e. non-retriable) invoke/service call,
- A non-blocking (i.e. parallel) invoke/send/throw, or
- Explicit dehydration, i.e. the checkpoint() call.

The presence of any of these in a synchronous process will force the engine to apply the same treatment (and therefore overhead) that applies to an asynchronous process. It also makes it likely that the process may not complete within the maximum wait time for synchronous processes, which may cause the caller to timeout and retry, further wasting resources on the engine. If there are many instances of such processes, there is a real danger of consuming too many worker threads in the engine and getting into a thread starvation situation. An excess of threads can also lead to a starvation of native memory.

If a synchronous process is transient in nature, consider disabling auditing and persistence for that process (composite) if it makes business sense to do so. This will save space and processing, however this should only be done if it is acceptable. If making this change, only faulted instances of these processes will be audited. This behavior is modified in Enterprise Manager by navigating to the composite and then pressing the Settings button and choosing the appropriate Composite Instance Audit Level setting.

Anti-pattern: Synchronous Fault Handling

Faults should always be handled asynchronously, so that the resources (transactions, memory, etc.) that the engine is using to run the process can be released after the fault has occurred. It is desirable to allow the engine to free these resources as quickly as possible so that they can be used for other work. Similarly, clients should be allowed to continue while the fault is handled, as handling a fault may take a relatively long time.

Synchronous fault handling makes it very difficult to rollback if a problem occurs. Insert an asynchronous point (like a JMS queue) between the failing logic and the fault handling logic. This allows the fault handling to be done on any (other) node in your cluster, and it makes it possible to rollback (to the queue) if the fault handler cannot complete.

Anti-pattern: Designing Processes with no Fault Handling

When designing (modeling) processes, consider what types of exceptions could occur and determine whether these need to be handled by the process. Include fault handling logic in process models. Do not rely solely on mechanisms outside of the process model, like the Fault Management Framework for example. Fault policies only apply to 'invoke' operations, e.g. a send task or invoke.

A common area where problems may occur is data associations. It is possible to get a **selectionFailure** when performing a data association. By default process instances are suspended on a data association error. If this occurs, it may cause the whole instance to be terminated and marked as not recoverable. Include handlers for this kind of fault in your process model, using either boundary events, or event sub-processes, or both.

Use transformation rather than assignment to deal with uninitialized data. This should not be disabled as it allows use of the Alter Flow feature to fix data issues.

A good practice is to include an event sub-process which catches all system and business exceptions in all process models.

Use of Metadata Services (MDS)

Always store common reusable artifacts in MDS. This means all of the WSDL and XSD files for example. Never include a concrete endpoint address in a process. This is essential to ensure that the

engine will always be able to start up, even if a remote service is not available – placing a concrete remote WSDL in a composite will cause the server to load that WSDL at startup of that composite. Storing artifacts in MDS also provides a single place to make changes and updates, and helps to ensure consistency.

Use of Sub-processes and Inter-Process Communication

The use of different kinds of sub-processes and inter-process communication can have an impact on performance. Consider the following:

- Minimize 'chattiness' by minimizing the number of conversations and messages in those conversations,
- Be aware that using an embedded sub-process or a reusable sub-process is less expensive in terms of resource and audit data production than calling another process,
- Whenever using a loop, set a completion condition so that the loop is able to terminate early if it is successful before all of the iterations have been completed,
- Only pass the minimum required amount of data to a sub-process – note that embedded sub-processes have access to their parent's data, so it is not necessary to send parent process data objects to them through their arguments,
- Be aware that exception propagation behavior varies with different styles of sub-process invocation, and
- Be aware that the parent process pauses while an embedded sub-process is executing – embedded sub-processes are not asynchronous.

Transformation or Data Association (BPEL Translate or Assign)

Always use data association (or assign activity in BPEL) for simple data transformations. The use of transformations (or the transform activity in BPEL) should be reserved for when a large number of fields need to be transformed and there is more complicated logic than just copying from source to destination, or for when the source variable is possibly uninitialized.

Data association (assign) should be used for the assignment of simple types and complex types where the whole type is copied. Where many children of a complex type need to be transformed, or where there is if/then logic is present, or where there are optional elements a transformation should be used instead.

Use of Loops

Avoid extensive use of loops. Use of large loops (i.e. loops with more than ten activities in them) to perform iterative processing tends to scale poorly with large numbers of instances and/or large variables.

Use of large loops can require the engine to keep large data objects in the JVM heap, perform a large number of database transactions for dehydration of state, and could also lead to garbage collection bottlenecks.

Each iteration of a multi-instance loop creates a new scope object. With each iteration the total allocated space grows exponentially ($1+2+3+..+n$). If a large loop contains a breakpoint activity (such as a wait), the entire scope tree will need to be persisted to the dehydration store, as well as any work items created since the last dehydration.

Do as much work as possible outside of the loop, before it starts, so that the minimum possible work can be actually done in the loop. This might mean manipulating variables before entering the loop for example.

If the loop is just being used to perform data manipulation and nothing else, attempt to eliminate the loop and use a transform activity (i.e. use XSLT) in its place to achieve the necessary data manipulation.

Dehydration and Transactions

The BPMN Service Engine uses transactions to control work and exception handling/retry boundaries. It is important to understand where these occur and the impact that they can have on the performance of your application.

Transactions

Understand where transaction boundaries occur, for example:

- Mid-process breakpoint activities (e.g. receive, onMessage, onAlarm, wait),
- A non-idempotent (i.e. non-retryable) invoke,
- A non-blocking (i.e. asynchronous) invoke,
- Explicit dehydration (in BPEL only), i.e. the checkpoint() call,
- An asynchronous queue (e.g. JMS), and
- A Timer Catch Event with duration of more than one second.

Explicit Dehydration

It is possible to ask the engine to perform an explicit dehydration in a BPEL process (using the **checkpoint()** function), however this should be used sparingly. BPMN processes do not have the ability to force explicit dehydration. However in BPMN, it is possible to force dehydration by modeling a timer catch event set to one second or longer.

Avoid Marshaling Overhead

Attempt to design the application and deployment environment to minimize the amount of marshaling of data that is necessary. Minimize inter-process communication.

Colocation

Consider co-locating components with other components that they communicate with. Doing so will not only avoid the network propagation delay, but will also avoid marshaling and un-marshaling the data as the engine is able to recognize that this is a local call and optimize it.

BPM/SOA clusters are homogenous, i.e. all components are deployed on all nodes, so nothing needs to be done to ensure colocation of composites. However, if using Oracle Service Bus or deploying web services, consider colocation with the BPM/SOA clusters to minimize latency.

Direct Bindings

Creating direct bindings (SOA-Direct, OSB-Direct, etc.) for components in addition to any other bindings, e.g. SOAP bindings allows the engine to avoid unnecessary marshaling. However, this may not necessarily improve overall performance and scalability. Calling services through OSB may provide more scalability as the OSB HTTP transport is capable of releasing threads to do other work while waiting for responses.

Asynchronous Service operations with callbacks

These types of invocations using message events or send activities, and the associated catching of the callback events can take longer than expected.

When an asynchronous service operation is invoked, the calling process does not wait for a response but instead continues to the next activity in the flow.

However if a subsequent activity in the calling process is either a message catch event or receive activity which invokes the service callback operation paired with an asynchronous event in the called process, then the calling process will wait at this activity/event until the calling process dehydrates.

Consider the example in Figure 3 which illustrates this scenario. Here we would expect the callback event from the called process to be caught immediately by the calling process and the calling process to end. Instead, because the 'write file' task takes a long time to complete, the 'catch callback event' is not released until the 'decide and wait' activity is reached. This can result in an unexpected, paused or slow execution or the calling process.

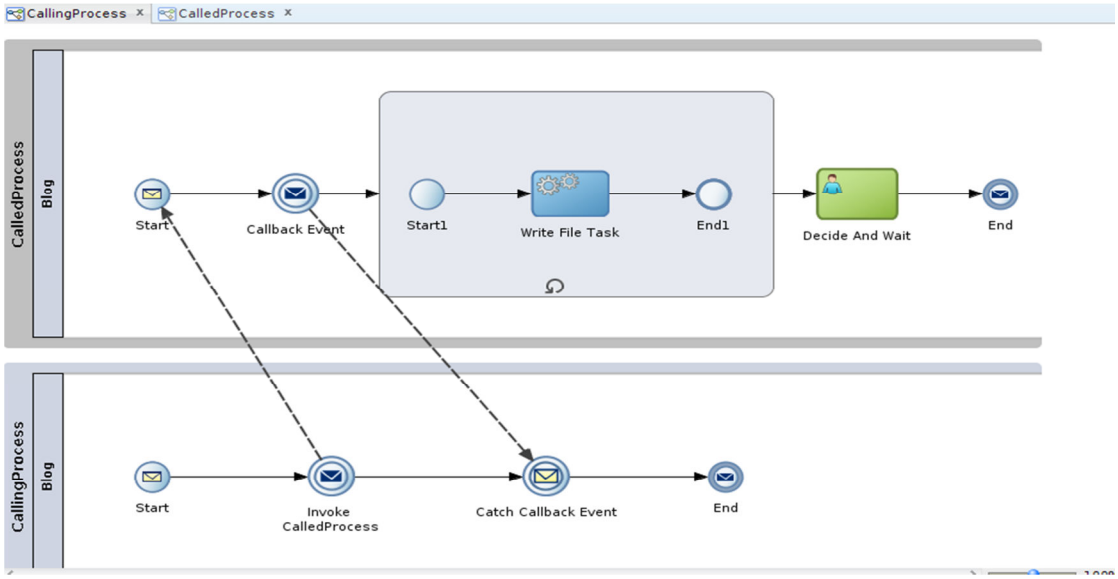


Figure 3. The calling process calls the called process asynchronously and catches the callback event.

Where there are business scenarios that require the called process to throw the callback event immediately and in a separate thread, then this can be achieved by setting the **nonBlockingInvoke** property in the **composite.xml** of the called process as follows:

```
<component name="CalledProcess">
  <implementation.bpmn src="processes/CalledProcess.bpmn"/>
  <property
    name="bpel.partnerLink.CalledProcess.service.nonBlockingInvoke"
    type="xs:string"
    many="true">true</property>
</component>
```

Use Caching

If it is necessary to work with a very large data set (larger than half of the heap size), consider obtaining a full use license for Coherence and use it to store this data set. To be completely clear: this means setting up a Coherence grid in which to store application data. This does not refer to any use of Coherence that is made by BPM internally, for example for coordination of deployment of composites in a cluster. Coherence usage for application data should be kept entirely separate from any such

internally used Coherence grids. Consider using features like Coherence Entry Processors to distribute processing throughout the grid (move the computation to the data, not the data to the computation).

Configuration of JTA Timeouts

When BPEL and/or BPMN processes are being invoked synchronously it is important that any timeouts are caught at the smallest (most local) scope where they can be handled and not propagated back to the global transaction unless necessary. In WebLogic Server, if a timeout has not been set explicitly, the default global transaction timeout of 30 seconds is used. BPM and BPEL EJB transaction timeouts are set to 300 seconds by default and the BPM **syncMaxWaitTime** configuration property is set to 45 seconds by default.

With these default settings, the default WebLogic Server global transaction timeout will occur before the other two and a message similar to the one shown below will be written to the WebLogic Server log file for the BPM managed server.

```
Transaction Rolledback.:
weblogic.transaction.internal.TimedOutException:
Transaction timed out
```

When this occurs, the application has no control over handling the timeout. In order to avoid this condition two things must be done: first establish the following relationship between the timeout values:

syncMaxWaitTime < BPM/BPEL EJB transaction timeout

Second, ensure that each component sets a timeout value explicitly and does not rely on the global default transaction timeout. Note that even if a component which has not set an explicit timeout calls one which has, if that timeout is larger than the global default it will not be used.

These two steps will ensure that timeouts are caught and can be handled at the smallest scope. If they cannot be handled at that scope, they will be bubbled up to the next scope, and so on.

Where to adjust these settings

The Global Transaction Timeout for the domain is set in the WebLogic Server console in the Configuration → JTA tab for the AdminServer.

The transaction timeout for BPM and BPEL EJBs are set indivually for each EJB on its Configuration page in the WebLogic Server console. The following EJBs need to have this setting checked and adjusted if necessary:

- BPELActivityManagerBean,
- BPELDeliveryBean,
- BPELDispatcherBean,
- BPELEngineBean,
- BPELFinderBean,
- BPELInstanceManagerBean,
- BPELProcessManagerBean,
- BPELSensorValuesBean,
- BPELServerManagerBean,
- BPMNActivityManagerBean,
- BPMNDeliveryBean,
- BPMNDispatcherBean,
- BPMNEngineBean,
- BPMNFinderBean,
- BPMNInstanceManagerBean,
- BPMNProcessManagerBean,
- BPMNSensorValuesBean, and
- BPMNServerManagerBean.

SyncMaxWaitTime is set in Enterprise Manager in the **BPELConfig:bpel** MBean, which is accessible from the SOA Administration → BPEL Properties menu.

Avoid Empty Elements in the Payload

Passing empty elements can add a significant overhead both in terms of marshaling and in terms of processing and data transformation logic.

Design processes and other components so that any optional data elements can be omitted from the payload and do not need to be passed as empty elements. Components should assume that any data element that is present is in fact populated with data and needs to be processed.

Handling Large Volumes

When handling large data volumes there are some important considerations to take into account when designing an 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. The remainder of this section discusses a number of these considerations.

Debatching

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

Transaction Boundaries and Dehydration

Be aware that when the BPMN Service Engine executes processes, it will use a number of database transactions for each process, possibly as many as one for each activity. Please refer to the **Database Tuning** section of this whitepaper for more information about this.

Attachment handling

If there is a need to handle large variables, files or attachments in a process, store the object in an external data store (like a content server, file system or database) and pass a URL or other pointer through the process in preference to passing the object itself. Passing the object through the process state (in a process data object for example) can cause a significant amount of storage consumption in the underlying database – process data objects are written to the database many times throughout the life of a process instance, and if the process contains a loop, you can quickly get into exponential storage growth. This effects the performance of all instances in the system and makes database maintenance more time consuming (since you have many more large BLOBs to deal with).

Complex decisions

If there is a need to make a complex decision, implement the decision in the rules engine rather than modeling a set of if/then logic in the process itself. Execution of this type of logic in the rules engine is much more efficient in terms of memory footprint, speed of execution and volume of audit data generated.

Variables

Scope must be considered carefully when designing a process. Consider where data objects are needed and declare them in the smallest relevant scope. It is possible to declare data objects at the process, sub-process and even activity level scope.

Avoid creating one large data object with all the data needed by all activities in the process. Instead create smaller, modular Business Object definitions that can be reused, are smaller to pass around and transform.

Avoid creating temporary variables for development purposes (whose later removal may be forgotten) and try to leverage the loggers in the activities instead.

Project Data Objects

Project Data Objects (as opposed to **Process** Data Objects) are useful where a definition of a Data Object should be shared across several processes in a project. Since they automatically map to flex field attributes, they should be used where the runtime value could be potentially used to provide additional filtering of tasks in the BPM Workspace (with custom views) or when using the Human Workflow API. These fields can then be analyzed and indexed further to provide improved performance. Additionally the value of the data object will be able to be obtained not only during execution of the process instance, but also afterwards, i.e. from completed BPMN process instances, which may be useful for building custom metric solutions.

Exception Handling

Exceptions can be handled in the process model itself or declaratively, in a fault policy file. The best practice recommendation is to handle system faults in fault policy (so that they can be changed by administrator without changing the model) and business exceptions in process model (so that they are part of the model).

Fault policies are attached to a composite. If a fault occurs during runtime in a process, the framework catches the fault and performs actions defined in the fault policy file. This provides better performance compared to exception handling in the process itself.

There are two options to handle exceptions in process model:

- Boundary event – These can be attached to the boundary of activities (tasks or sub-processes) to capture events thrown by activities. The scope of a boundary event is the activity or the sub-process to which it is attached.
- Event sub-process – These are used as inline event handlers. They are similar to boundary events except that the event handlers run inside the context of the original activity. They cannot have outgoing or incoming sequence flows. The event sub-process is contained within a process (or an embedded sub-process) and can be activated any time during the life time of

the process (or the embedded sub-process) containing it. They are not attached to a task. They are triggered by (i.e. catch) start error events.

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 performing adequate testing. Many performance problems will not show up in development and only become apparent when applications are moved to a production-like environment.

Conduct testing of all major components of the system before production cutover, including:

- End-to-end functional testing,
- Load testing, and
- Longevity testing.

Test that the application functions correctly in the production environment, that it functions correctly in a cluster (if using one) and that it can handle sustained load.

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

If project timelines prevent adequate testing before production cutover, 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. Identify these risks to the project manager and/or sponsor as early as possible so that they can be understood and mitigated.

Each of the following recommendations should be observed during testing. It is important to note that most of these will require some specialist skills to collect and interpret the results. Identify one or more individuals who can learn these skills, as they will be required each time additional users or components are released 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,
- VisualVM or similar tools should be used to help identify issues at the JVM level, e.g. garbage collection efficiency, memory leaks, thread contention, etc. If using JRockit, it is highly recommended that you conduct a JRA recording during the load testing, and
- Oracle Database Automatic Workload Repository (AWR) and System Performance (SAR) reports should be run and the output analyzed by an experienced DBA to help identify any performance issues at the database level. 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. Configure the database to store BLOBS in separate tablespaces.

Process Engine Tuning

The BPMN Service Engine (which runs BPMN processes) can be tuned in various ways to improve performance. The optimal settings during development of an application may be quite different to those for deployment of that application into a production environment. This section describes the major tuning options for the BPMN Service Engine.

Tune the Instance Numbering Block Size

Instance numbers are allocated in blocks. The default block size is 1000. If there are more than 200 instances created per minute, consider increasing the size of the block. The setting that controls this is called **AuditKeyExtents** and is found in the BPMN Service Engine properties, (which are accessed from Enterprise Manager → soa-infra → SOA Infrastructure → SOA Administration → BPMN Service Engine → Properties).

Stop Unnecessary Components

When installation is completed, there will be several components that are configured to start up by default which may not actually be needed. If these components are not being used, stop them (in the WebLogic Server console). When they are stopped, they will not restart, even after a server restart, until they are explicitly restarted.

You should consider stopping:

COMPONENTS TO CONSIDER STOPPING

COMPONENT	NOTES
worklistapp	Included for backward compatibility. Use OracleBPMWorkspace instead.
composer	Included for backward compatibility. Use BPMComposer instead.
b2bui	Turn off if not using B2B functionality.
OracleAppsAdapter	Turn off if not using the Oracle Applications Adapter, i.e. the E-Business Suite adapter.
OracleAqAdapter	Turn off if not using Oracle AQ.
MQSeriesAdapter	Turn off if not using WebSphere MQ.
OracleBAMAdapter	Turn off if not using Oracle BAM.

It is reasonable to expect that stopping these components would reduce the memory usage by around 100 to 120 MB.

Note that **composer** has some functionality for editing Domain Value Maps that **BPMComposer** does not include. If you are using DVM's you may wish to keep **composer** active.

In order to disable the loading of B2B at server startup, it is also necessary to set the **b2b.donot_initialize** property to true. This property must first be defined (in Enterprise Manager → soa-infra → SOA Infrastructure → SOA Administration → B2B Server Properties → More B2B Configuration Properties… → Operations). The property name is **b2b.donot_initialize** and the value is **true**. The managed servers must be restarted after defining this property.

Disable DBMS job to refresh B2B Materialized View

If B2B is not being used, disable the DBMS job that refreshes the B2B materialized view. This job runs every minute by default.

To find the job, use a query like the one below. Update the schema user to match the environment:

```
select
  job
, schema_user
, broken
, what
, interval
from
  dba_jobs
where
  schema_user = "DEV_SOAINFRA"
```

Look for the job which contains something like the following in the WHAT column. Note that it may be different if a different schema prefix was used in the environment.

```
dbms_refresh.refresh('"DEV_SOAINFRA"."B2B_SYSTEM_MV"');
```

To remove the job, take note of the job number from the JOB column and then use a command like the following, substituting in the correct job number in place of '24':

```
begin
  dbms_job.remove(24);
end;
```

Alternatively, alter the materialized view to be refreshed on demand, using a command similar to this:

```
alter materialized view dev_soainfra.b2b_system_mv refresh on demand;
```

Data Management – Purging

Management of the SOAINFRA database schema is one of the most important considerations for ensuring good overall performance of BPM. Review the information on purging in the Database Tuning section of this whitepaper, and review the other references listed in that section, for more information.

Production and Development Modes

Run production servers with both WebLogic Server and the SOA/BPM Infrastructure set to 'production' mode. Setting the mode to 'development' can have a significant negative impact on performance. If it is ever necessary to change the setting for the SOA/BPM Infrastructure (e.g., if trying to capture some additional logging information to help diagnose an issue), then ensure that the setting is changed back to 'production' as soon as practical. It is not necessary to change WebLogic Server to 'development' mode for debugging purposes.

For the SOA/BPM Infrastructure, ensure that the Audit Level is set to 'production' in the 'SOA Infrastructure Common Properties' in Enterprise Manager as shown in Figure 4. Additionally, set 'Capture Composite Instance State' and 'Payload Validation' to off (i.e. uncheck them) in production environments.

Note that it may sometimes be desirable to retain instance tracking in Enterprise Manager. In this case, leave the 'Capture Composite Instance State' set to on (i.e. checked) and be aware that there will be a small overhead.

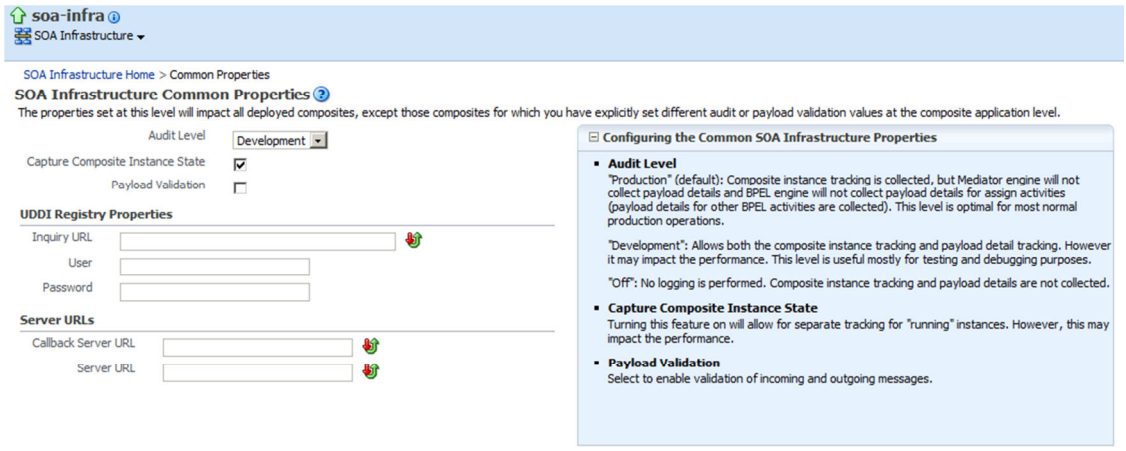


Figure 4. SOA Infrastructure Common Properties

Payload Validation

This setting should only be enabled in test and development environments under normal circumstances. Consider enabling this in production for a short period of time if you are attempting to collect diagnostic information.

BPMN and BPEL Engine Configuration

Configure the number of threads used for various purposes in the BPEL and BPMN engines. Figure 5 shows an example of the settings for the BPMN engine. Ensure familiarity with these settings and what they do.

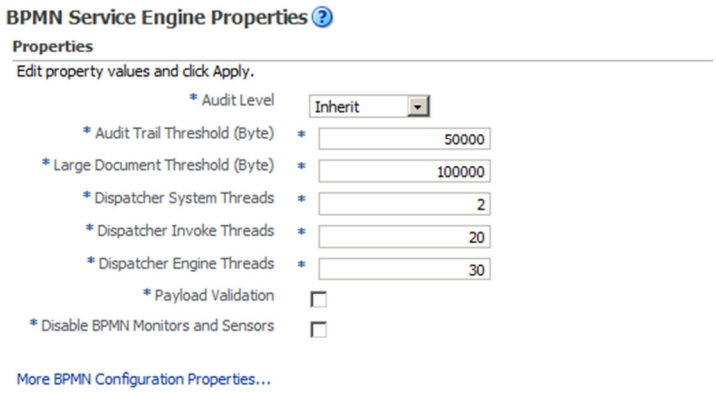


Figure 5. BPMN Service Engine Properties

Here is a list of the most important settings and some guidance on the impact of changing them. Click on the 'More BPMN Configuration Properties' link and review the details of the other settings that are available.

BPMN SERVICE ENGINE PROPERTIES

PROPERTY	NOTES
Dispatcher Invoke Threads	Number of threads available to process invocation messages, this controls how many service invocations can occur at any one time. Higher settings will allow more concurrency.
Dispatcher Engine Threads	Number of threads available to process engine messages. Higher settings will allow more concurrency.
Dispatcher System Threads	Number of threads available to process system messages. Higher settings will allow more concurrency.

BPMN SERVICE ENGINE PROPERTIES

PROPERTY	NOTES
Audit Level	Controls the granularity of audit trail messages that are logged. Higher settings will use more memory and increase database activity.
Payload Validation	Controls whether the engine will validate incoming and outgoing payload XML documents. Turning this on will increase CPU utilization.
Large Document Threshold	The maximum size of a variable that will be stored in the normal database tables. If a variable is larger than this value, it will be stored in a separate table for large documents. Higher settings may impact database performance.
One Way Delivery Policy	Controls whether one way messages are persisted and whether such persistence is synchronous with delivery. This setting will affect response times and CPU utilization.

A Note on Heap Utilization When Adjusting Threads

Be aware that increasing the number of threads in the engine will almost certainly also increase the heap utilization. Ensure that the impact of these changes is monitored and there is enough heap for the number of threads configured. If there is available memory, increase the heap size when increasing the number of threads in the engine.

Consider that a larger heap will probably mean longer garbage collection times. If changing the thread count or heap size, carefully consider the garbage collection algorithms and the new/old ratio.

On 32-bit JVMs, increasing the number of threads will consume more native memory (which is limited at 2 GB on 32-bit Windows operating systems), so the maximum heap size (**-Xmx**) may need to be reduced to allow for this.

Data Source Configuration

A number of data sources are used by the SOA/BPM Infrastructure to connect to the database server(s). There are a number of options that should be tuned on each of these data sources. These settings are found in the 'Connection Pool' tab of the Data Source 'Configuration' page in the WebLogic console. Click on the 'Advanced' link to see all of these options.

Initial and Maximum Connection Pool Capacity

Ensure that the 'Initial capacity' and 'Maximum capacity' are appropriate based on concurrent access to the database. 'Appropriate' means that the total across all data sources should be less than the maximum number of connections allowed by the database. Initial capability should be large enough to

handle a reasonable load, so that it is not necessary to obtain new connections when processing load increases.

Connection Creation Retry Frequency

Ensure that the 'connection creation retry frequency' is set to a non-zero (number of seconds) value. This will prevent the system from creating many connections in quick succession, which would more than likely make the problem worse rather than better. A small positive number (1-3) is probably satisfactory, however this should be monitored to determine the optimum value for a specific environment.

XA Connection Timeout

Enable 'XA Connection timeout' and set the value of 'XA Connection Timeout' to 0 (seconds). This will prevent BPM from holding on to connections for an extended period of time before releasing them, and in turn will help to prevent connection starvation (running out of connections).

Shrinking

Consider disabling shrinking to avoid repeated opening and closing of connections. Alternatively, set the shrink period to a large value.

If it is expected that the environment will have infrequent spikes and very low normal usage, it is safe to ignore this setting.

Statement Cache

Consider the statement cache hit/miss ratio. There is a tradeoff between memory consumption and statement cache (the number of statements cached per connection). If hit/miss ratio is poor (low percentage), then consider increasing the size of statement cache. Information on connections displayed in the WebLogic Server console (e.g. maximum connections, maximum wait time, etc.) should be used to understand JDBC behavior during performance test runs.

Directory Configuration

If LDAP is being used for either the user or policy stores, and especially when using it for the user store, there are some LDAP optimizations that should be made, and which are described in this section.

Suboptimal configuration of LDAP can result in slow login times. It is important to understand how LDAP configuration can impact on the performance of BPM. When a user logs in to BPM, the

container (WebLogic) will call the LDAP provider for authentication. After successful authentication, the SOA verification service will then call the LDAP provider to obtain group/role and permission information and will create a BPM user object. This second interaction can take a significant amount of time – tens of seconds – and the aim is to avoid that delay.

Ensure all LDAP patches are applied

Ensure that all of the available LDAP performance patches applied. First, upgrade environments to BPM 11.1.1.6 if not already on this release. This release includes many important LDAP fixes. Then apply patches 13454820 and 13791927.

LDAP Tuning

Applying the patches alone will not be enough to ensure the best performance, it is also necessary to carry out some tuning. The following tuning actions are recommended:

- Ensure that there are indexes on attributes in the searchable attribute list:
    - The recommended searchable attribute list is: **cn**, **sn**, **givenName**, **uid**, **manager**, **title**, **mail**, and **telephoneNumber**,
    - The presence of an index on **manager** is very important, as BPM loads the management hierarchy from the LDAP when you login,
    - For more information, you should refer to http://download.oracle.com/docs/cd/B31017_01/integrate.1013/b28982/service_config.htm#sthref306
- Keep the **search base** as narrow as possible. Consider creating a separate tree in your LDAP for BPM users if this is practical/possible. For more information, please review http://docs.oracle.com/cd/E21764_01/core.1111/e10108/human.htm#BABJCDDA
- If using Active Directory, consider this additional tuning:
    - If using nested groups, enable token groups, but measure the impact of this change. If it has a negative impact, add predicates to narrow queries to just those groups that are actually used for authorization. The use of nested groups has been observed to produce quite different results in different environments, so it is not possible to predict if it will have a positive or negative effect,
    - Enable group caching.
- Specify the **maximum group membership** search level to a reasonable number, like 3 for example,
- Consider following LDAP Referrals, if referrals are enabled in the LDAP server,
- If the LDAP server supports caching, leverage caching at the LDAP server level,

- For more information on group membership cache and connection pool size, please refer to http://docs.oracle.com/cd/E17904_01/web.1111/e13707/atn.htm#BABFHHGE

What to do if you still have slow LDAP performance

If these recommendations have been exhausted and LDAP performance is still unacceptable, consider the following additional actions:

- Trace LDAP calls and timings in order to find slow queries and consider creating indexes to make those queries faster. Use a tool like Wireshark to trace the LDAP calls that the BPM server is making,
- Collect more information by enabling performance profiling using the following settings:

```
-Doracle.bpel.services.perf.enabled=true
-Doracle.bpel.services.perf.dir=$ORACLE_HOME/perf/
-Doracle.bpel.services.perf.file=perf.log
```

Note: The trailing slash at the end of the second setting is required.

- Run **oidstats.sql** to collect statistics on the OID database,
- Ensure the OID database is on different disk(s) than the SOA database, and
- Use AWR on the OID database to understand its performance.

Find more details on LDAP performance profiling in the following document:
https://support.us.oracle.com/oip/faces/secure/km/DocumentDisplay.jspx?id=1267753.1

Other options

There may be instances where it is not possible (e.g. company policy) to change the LDAP configuration. In such instances a viable alternative could be to create a copy of the LDAP users/groups needed by BPM in a smaller, lightweight BPM-specific LDAP instance. This would allow this instance to tuned independently and remove the overhead of searching large user trees of other applications. This can increase the performance of the LDAP calls considerably without having to impact the global configuration of the corporate LDAP. For users of Active Directory, LDS or ADAM could be such an alternative. It should be noted however that automated ways to sync user credentials should also be employed, where required.

Audit Trail and Log Configuration

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.

API Considerations

When using the **TaskQueryService.queryTasks()** API, it is possible to adversely affect performance by:

- specifying **Predicate** objects that are not selective enough,
- specifying large page sizes, or
- specifying expensive **OptionalInfo** options.

When using the BPM API, you should always cache the **WorkflowContext** object (for each user) and reuse it for each subsequent API call by that user. Repeatedly calling **authenticate()**, **authenticateOnBehalfOf()** or **createContext()** is very expensive by comparison.

A note on task assignment distribution

The distribution of tasks/processes across users can influence overall performance. When assignment is skewed heavily to a single assignee (e.g., all the tasks in the system assigned to a single group), this can cause performance issues. This is unlikely to occur in a production system, but it could easily occur in a test environment. This should be considered when designing test cases.

Analytics

Analytics preferences can be configured at both the BPM project and process level. These settings include the configuration on when the BPMN Service Engine should take process-sampling points and where to publish the data. The default is to sample for interactive activities of a process and publish the data to the BPM cube tables in the database.

If no analytics information is required for the project, switch off the sampling, i.e. select 'Do not generate', and data targets.

Be aware that while the analytics preferences can be set for individual BPMN processes, the default is to inherit then from the BPM project preferences and then in turn from the BPMN Service Engine settings.

WebLogic Server Tuning

The Performance and Tuning Guide (see link below) provides extensive information about best practices for configuring and tuning the WebLogic Server and JVM. Review this document, at least Chapter 2 ('Top Performance Areas'), and implement the recommendations.

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

Transaction Logs

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

JVM Tuning

BPM is supported on various JVMs. The choice of JVM has an impact on performance. The general rule of thumb for BPM is to use HotSpot in preference to other JVMs. For SPARC, we have found that most applications will get better performance with HotSpot. Use the latest stable release of the JVM that is available. Use the same JVM and the same version of that JVM in all environments.

The second main consideration, if 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, consider using the **UseCompressedOops** (or equivalent) JVM setting, which will cause the 64-bit JVM to use less memory for references.

It is reasonable to expect the use of **UseCompressedOops** to reduce memory usage by something in the order of 10%.

As a general rule of thumb, 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 an application fits into that category, and it does not need to keep as much data in memory in any one JVM, 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 to test each application and tune the JVM. This should be an iterative process to find the best configuration for the specific environment and workload.

When tuning your JVM(s) consider sizing of the heap and permanent generation (in HotSpot) as well as the garbage collection algorithm selection and configuration. Remember that the heap is not the only memory that is used by the JVM.

Recommended Settings

There are some basic JVM settings that are recommended in most cases. Consider using these settings in all of your environments. The table below summarizes the recommended settings, which are discussed in the following section.

Start with a small set of options which are as simple as possible, for example just **Xms**, **Xmx** and possibly the garbage collection algorithm. This will allow the JVM to make decisions about its behavior. Add additional settings only if there is a problem seen in the collected JVM statistics.

In the table below, 'recommended' indicates settings the settings that are often found to be in use after this process of tuning.

Note that **XX** settings are by definition experimental and are subject to change without notice.

RECOMMENDED JVM SETTINGS

HOTSPOT	JROCKIT	PURPOSE
-XX:PrintGCTimeStamps -XX:PrintGCDetails -Xloggc:filename.log	-XX:PrintGCTimeStamps -XX:PrintGCDetails -Xloggc:filename.log	(Recommended) Enable verbose garbage collection logging in production. Logs will be written in to the file named in the –Xloggc setting.
-XX:+HeapDumpOnOutOfMemoryError	-XX:+HeapDumpOnOutOfMemoryError	(Recommended) request that the JVM dump the heap if an OutOfMemoryException occurs.
-server	n/a	(Recommended) Tell the JVM to run in 'server' mode.
-Xms -Xmx	-Xms -Xmx	(Recommended) Set the initial and maximum heap size. Generally set them the same to avoid the overhead of heap size adjustment.
-XX:PermSize -XX:MaxPermSize	n/a	(Recommended) Set the initial and maximum size of the permanent generation. Generally set them the same to avoid the overhead of permanent generation size adjustment.
-XX:+UseCompressedOops (default in JDK 1.6.0_23+)	-XXcompressedRefs	(Optional) Tell a 64-bit JVM to use less memory for references.
-XX:+UseParallelOldGC	-XgcPrio:throughput	(Optional) Tells the JVM to optimize the garbage collection for best throughput – will run until the heap is nearly full, and then stop all threads to garbage collect. The alternative is to optimize for latency.
n/a	-XXtlaSize:min=1k,preferred=512k	(Optional) Thread local allocation size.
-XX:+CMSClassUnloadingEnabled -XX:+UseConcMarkSweepGC	-XgcPrio:pausetime or -XgcPrio:deterministic	(Optional) If you are running ADF applications – like Human Task user interfaces on the same managed server as BPM, you should consider using the Concurrent Garbage Collection algorithm. This has been

RECOMMENDED JVM SETTINGS		
HOTSPOT	**JROCKIT**	**PURPOSE**
		shown to give better performance in this particular scenario.

Garbage collection logging is very useful when attempting to tune a JVM. The overhead of collecting this information is very low. Turn on garbage collection logging in all environments including production. If there is a need to do some performance tuning, it is helpful to have to have actual garbage collection data from the production environment. Leaving garbage collection logging on all the time will mean that the data will be available if it is ever needed. If a problem occurs, it will not be necessary to attempt to reproduce it in order to capture garbage collection data, and there are tools that will parse the garbage collection log data to produce information about the garbage collection behavior during the test.

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

If using the HotSpot JVM, 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. Generally the initial and maximum heap size should be set the same to avoid this additional overhead. This is also appropriate for heap sizes under 2GB, but is of reduced value in those environments compared to environments with larger heaps.

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

If running in a 64-bit environment and the heap is approaching the size of the available memory, 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 the environment, conduct some testing to analyze the impact of using this option. Note that JRockit will use compressed references by default in some circumstances.

Note that **UseCompressedOops** is supported and enabled by default in Java SE 6u23 and later. In Java SE 7, **UseCompressedOops** is the default for 64-bit JVM processes when **Xmx** isn't specified or is less than 32GB. For JDK 6 before the 6u23 release, use the **-XX:+UseCompressedOops** flag to enable the feature.

Garbage Collection Algorithm

The recommended garbage collection setting for SOA/BPM is **UseParallelOldGC** which will cause the JVM to use the Parallel Scavenger collector in the young generation and the Parallel Old collector in the tenured generation.

If running ADF applications (human task user interfaces) on the same managed server as BPM is running on, use the concurrent garbage collector (**UseConcMarkSweepGC**).

A new garbage collector, known as G1, has been introduced in JDK 7. Not enough testing has been done with this new garbage collector in order to give a recommendation at the date of writing this paper.

Database Tuning

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

What to tell the DBA about the SOAINFRA Schema

Often a DBA will want some basic information about which objects are important and what makes the database grow, both in terms of the number of rows, and in terms of how big each of those rows is. This section presents the information that has been found to be most useful to provide to DBAs in order to help them understand how the database is likely to behave when running BPM.

All BPM metadata is stored in the SOAINFRA schema. BPM instance state is stored across the SOA and Human Workflow (HWF) tables:

- SOA tables are used for message dispatch, state management, messages, payload and headers, pending requests (e.g. callbacks for timers), etc., and
- HWF tables are used for work items, the query framework, comments, attachments, callbacks for tasks, etc.

The diagram below presents a subset of the SOAINFRA schema, highlighting those tables that are important for BPM.

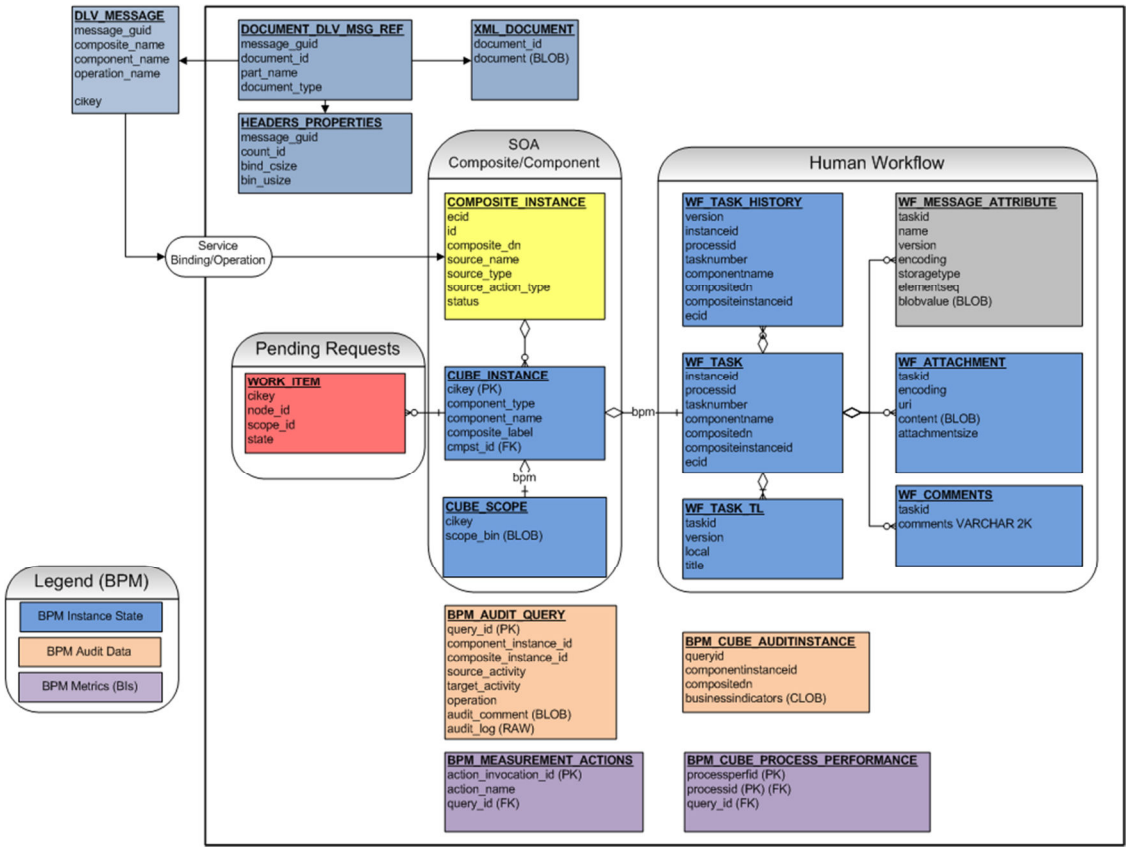


Figure 6. Tables used to store state in BPM (not a complete ER diagram)

Core State Tables

The table below lists the core database tables that are used to store BPM state, and what kind of information is stored in each of those tables.

TABLE	WHAT IT HOLDS
CUBE_INSTANCE	Represents process
CUBE_SCOPE	Instance state as BLOB
WF_TASK	Task Instance and Link to Process Comments and Attachments
WF_TASK_TL	Title translation
WF_MESSAGE_ATTRIBUTE	Task payload
WF_ATTACHMENT	Task and Process Attachments

TABLE	WHAT IT HOLDS
WF_COMMENTS	Task and Process Comments
WF_TASK_HISTORY	Change Events
WF_ASSIGNEE	Assignees for each task/process instance - one row per assignee/per instance

What creates rows

It is important to understand what kinds of operations will result in rows being created in the database. The key operations to be aware of are:

- Creating process instances results in inserts into core state tables,
- Updates to task payload,
- Asynchronous operations cause storage of message payload and "bookmark" in XML_DOCUMENT, WORK_ITEM, DLV_SUBCRIPTION, etc., and
- Audit records are written for entry and exit of every scope and model element.

Additionally, there are some important "multipliers" to be aware of. These result in the creation of multiple rows:

- Process scopes (one row per scope created in WFTASK),
- Task assignments (one row per assignee created in WFASSIGNEE), and
- Task payload design (one row per element created in WFMESSAGEATTRIBUTE).

Being aware of these considerations will also help you to design your processes to minimize the database overhead.

A note on WF_ASSIGNEE

The data volume per row is not as significant as WF_TASK and WF_TASK_HISTORY, but nonetheless, WF_ASSIGNEE is a key table in the queries for the task list and process tracking lists in the BPM Workspace. As WF_ASSIGNEE can potentially have multiple rows per instance, the number of rows in the table can grow large.

Core Audit Tables

Enter and exit events are recorded for each scope and element. Events are stored in BPM_AUDIT_QUERY.

For example, one instance of the following process model produces twenty audit events:

- 2 scopes (process, sub-process) and 8 BPMN elements, and
- 10 total events × 2 (for enter and exit) = 20 events per instance.

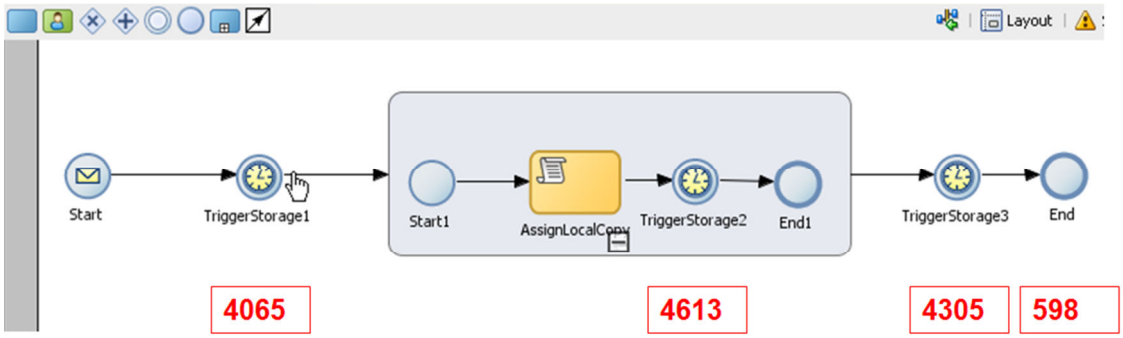


Figure 7. Example Process

What causes changes to row sizes

Just as it is important to understand what operations create more rows, it is also important to know what will makes those rows larger:

- Hydration points causes state to be written, e.g. asynchronous activities or events,
- Transition to a new scope takes up additional space, however leaving the scope releases the state stored for that scope, and
- Instance state is released when a process instance completes.

As an example, consider the process above. With a 166 byte payload, the red boxes indicate the payload size that would be stored in CUBE_SCOPE at various times in the execution of the process instance.

A note on running statistics

It has been observed that high CPU usage on the database can occur when there are many concurrent users logging into the BPM Workspace and/or when there are many tasks created in a short period of time. The typical issue here is that database statistics have not been collected resulting in a sub-optimal plan being generated by the Oracle cost-based optimizer. It is recommended that DBAs run **anaylze table** periodically (at least once a day initially and weekly after that).

```
analyze table wftask compute statistics;
analyze table wfassignee compute statistics;
```

How to Query Size of Process Instance State

The following query can be used to find the size of the state data for a process instance. This information is stored in the database in a compressed format, so the uncompressed numbers are more useful for making high level comparisons about parsing times and compression overheads. The compressed numbers are useful for knowing how much storage space is being used in the database.

```
select
  count(*),
  min(scope_csize) minc,  -- scope_csize is the compressed size
  max(scope_csize) maxc,
  avg(scope_csize) avgc,
  min(scope_usize) minu,  -- scope_usize is the uncompressed size
  max(scope_usize) maxu,
  avg(scope_usize) avgu
from
  cube_instance
```

Purging

Purging, partitioning and general management of the SOAINFRA database has proven to be a major factor in ensuring good performance in production deployments of BPM (and SOA).

Failure to properly manage the database is the single most significant factor causing performance issues that has been observed. It can impact performance of runtime SQL queries, asynchronous routing rules and asynchronous BPEL processes. It can slow down console access when querying for auditing data. There can be a very large overhead when purging data – longer maintenance windows are required, defragmentation of the database can take a long time. If retention periods are long, significant amounts of disk space are required and tables with BLOB columns grow to take up a majority of the database space.

Extensive work has been done in this area to provide patches and practical guidance on how to manage your SOAINFRA database. In addition to ensuring environments are on the latest release and have all the appropriate patches installed, review the 'SOA 11g Database Growth Management Strategy' whitepaper and implement its recommendations. This whitepaper is available at http://www.oracle.com/technetwork/database/features/availability/soa11gstrategy-1508335.pdf

Partitioning

From release 11.1.1.6, the Repository Creation Utility (RCU) can automatically set up database partitions by default. This helps to improve performance of operations like searching for process instances or tasks when there is a large amount of historical data in the SOAINFRA schema. Use the partitioning settings provided, and monitor them and adjust them as necessary based on the volume of data in your environment.

Automatic Workload Repository

In order to continue to get good performance from the BPM server, it is necessary to periodically check the database – the one storing the SOAINFRA schema – to see if there are any performance issues there. Keep redoing this, as the workload changes and the usage of space in the database changes. Depending on the volume of traffic going through the system, consider tuning the database every week or every month for example.

Tuning an Oracle database is a specialist task. It requires a highly trained and experienced DBA to do it well. If there is not such a person on staff, consider hiring one or contracting with a consultant or service provider who can provide these skills.

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 it is necessary to license the Database Tuning option from Oracle.

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

Here are some specific areas to check in a database that is supporting BPM/SOA:

SPECIFIC INFORMATION TO CHECK IN AWR REPORTS FOR SOA/BPM

DETAIL	NOTES
Redo logs	There will normally be a lot of redo activity on the SOA database. Ensure that the redo logs are large enough. Place them on a different disk to the database files. Check the number of log switches, one every twenty minutes is ideal, more than this is too high and indicates that the redo logs should be made larger to reduce the number of switches.
Parsing	Check the hard parsing amount. It should be zero. If it is not, this indicates that the SGA is probably too small, increase the size of SGA and test again. Hard parsing is caused by use of literals in SQL (as opposed to bind variables). If the queries in question are customer developed, e.g. in a database adapter, then change them to use bind variables.

SPECIFIC INFORMATION TO CHECK IN AWR REPORTS FOR SOA/BPM

DETAIL	NOTES
SGA	Check the buffer hit and library cache hit percentages. They should be 100%, if not increase the size of SGA.
MEMORY_TARGET	Do not use this setting. Have the DBA tune the memory manually instead. This will result in a better tuned database. Start with 60% of physical memory allocated to SGA and 20% to PGA.
AUDIT_TRAIL	Do not use this setting.
Top 5	Check the average wait times. Anything over 5ms indicates a problem. If there are database CPU events in the Top 5, this indicates that SGA is too small. There may be missing indexes. Check the optimizer statistics*.
Database file sequential/scattered read	These indicate time spent doing table scans and index scans (respectively). If these are high, move data files to more disks to reduce disk I/O contention, or move them to faster disks.
Enqueue high watermark	This indicates hardware contention that occurs when there are multiple users inserting into LOB segments at once while the database is trying to reclaim unused space. Enable secure files to improve LOB performance (SECURE_FILES=ALWAYS).

* To gather optimizer statistics, use a command similar to the following, substituting in the correct schema name for the environment in place of **DEV_MDS**:

```
exec dbms_stats.gather_schema_stats(
  ownname => 'DEV_MDS',
  options => 'GATHER AUTO')
```

Missing Indexes

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

One important example of missing indexes is Flex Field columns in WF_TASK. These are not indexed by default. When planning to use a Flex Field column for searching process instances or tasks, it is advisable to create an index on that column.

Use Separate Tablespaces for BLOBs

If large BLOBs are being stored in the database, create a separate tablespace for the BLOBs. If there are different types of storage available, consider where to place this tablespace on the storage farm in order to get optimum performance of queries. Discuss this with the Database Administrator.

Use secure files to improve LOB performance. It is enabled using the setting **SECURE_FILES=ALWAYS**, but must be done before defining objects, i.e. before running RCU. If objects are already defined, it is possible to use the PL/SQL online redefinition package to migrate existing LOB data into secure files. Talk to the Database Administrator about how to do this.

Redo Logs

Ensure that the database redo logs are placed on different disks than the database files and that they are large enough to support the environment. See the AWR section above for more details.

Tools to have in a Performance Kit Bag

This section presents a list of tools that are recommend to be familiar with and available when performance tuning. Since performance tuning is often done at the same time as investigating a problem, tools that are useful when you need to analyze a problem to determine its root cause are also included, as well as those that are useful for performance tuning work. The tools recommend here are the same tools that Oracle's performance specialists use internally.

TOOL	COMMENTS
Thread Dump	Know how to make WebLogic take a thread dump. You are often going to want a number of thread dumps over a period of time. On Unix-based systems, **kill -3 <pid>** is the best. You can also use **jstack** for HotSpot or **jrcmd <pid> print_threads** for JRockit. Thread dumps are useful for seeing what is happening inside your server and for identifying resource contention issues.
Heap Dump	Know how to make WebLogic take a heap dump. You can also use **jmap** for HotSpot or **jrcmd <pid> hprofdump filename=dump.hprof** for JRockit R28 and newer. Heap dumps are useful for seeing how memory is being used, for analyzing memory problems, troubleshooting problems that require detailed information from objects in memory, and also for JVM tuning.
MAT, YourKit or simliar	Memory Analyzer Tool. MAT is useful for reading and analyzing heap dumps, especially for analyzing memory problems. Be aware that it is necessary to have more memory in MAT's heap than the size of the heap dump to be analyzed. YourKit is often more convenient for reading large heap dumps, however it can be more complicated to use than MAT.
VisualVM	This is part of the HotSpot JDK. This is a good tool for watching a JVM in real time and seeing thread and memory usage. It has a plugin called VisualGC that allows you to watch the garbage collection in real time. This is useful for JVM tuning and sizing.
AWR	Application Workload Repository. This is part of the Oracle database. It is useful for understanding what is happening in your database, and for database tuning. This is a tool that should be used on an ongoing basis, not just a one-off tuning exercise.
ThreadLogic	ThreadLogic builds upon the popular TDA (Thread Dump Analyzer) by adding logic for common patterns found in application servers. ThreadLogic is able to parse thread dumps and provide advice based on predefined and externally defined patterns. You can find more info on ThreadLogic at

TOOL	COMMENTS
	http://java.net/projects/ThreadLogic.
Wireshark	Allows you to monitor packets travelling over the network – a fairly low level tool. Very useful for tuning LDAP performance issues.
OATS/LoadRunner/JMeter	Load testing tools. Select a tool that can be used to record (or script) load tests and to replay these against your server for load testing. It does not matter so much which particular tool is selected, as it does that one is selected. If it is necessary to test any ADF user interfaces then ensure the tool that can handle HTTP headers, substitution of parameters in the HTTP query string and cookies.
JRMC + WLDF plugin	JRockit Mission Control Useful for observing, recording and analyzing JVM performance. The WebLogic Tabs plugin allows you to trace ECIDs.
WLDF	WebLogic Diagnostic Framework. Useful for collecting and analyzing low level diagnostic information from WebLogic Server.

Conclusion

This paper has presented a collection of recommendations and options. It is important to remember that there is no globally relevant set of performance tuning recommendations for BPM, as the applications that are built and run on top of BPM vary widely. It will be necessary to test these recommendations to determine which ones are appropriate in a given environment.

Also, remember that performance tuning is not a one off exercise. It needs to be repeated regularly, and especially when there are significant changes to the workload profile and when new applications are deployed.

Performance tuning is an iterative process and must be carried out with care to achieve the best results.

Performance Tuning for Oracle Business Process Management Suite 11g
March 20132

Editor: Mark Nelson. Contibuting Authors: Vikas Anand, Deepak Arora, Heidi Buelow, Christopher Karl Chan, Manoj Das, Pete Farkas, Mark Foster, Simone Geib, Sabha Parameswaran, David Read, Derek Sharpe, Sushil Shukla, Kavitha Srinivasan, Meera Srinivasan, Shumin Zhao. Reviewers: Partricio Barletta, Kim LiChong, Andrew Dorman, Ralf Mueller, Bhagat Nainani, Robert Patrick, David Read, Will Stallard.

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