An Oracle White Paper
December 2012

Oracle AIA 11g Performance Tuning on Oracle’s SPARC T4 Servers
A Guide for Tuning Pre-Built Integrations on Oracle AIA 11g
Introduction ........................................................................................................... 1
Guidelines ............................................................................................................. 3
  Performance Testing Guidelines ................................................................. 3
  Tuning Knobs Guidelines ............................................................................. 6
    Operating System ......................................................................................... 6
    JVM ............................................................................................................... 6
      Tuning Hotspot JVM ............................................................................... 6
      Tuning the Memory Footprint .................................................................. 6
      Enabling GC Logs .................................................................................... 7
      Tuning Garbage Collections .................................................................... 8
      Tuning Other Parameters ......................................................................... 9
Database ............................................................................................................... 9
  Tuning Oracle Weblogic Server ................................................................. 11
    JTA Transaction Timeout ......................................................................... 11
    JDBC Connection Pools ............................................................................ 12
  Tuning Oracle SOA Infrastructure ............................................................. 12
    Tuning Oracle JMS Adapters .................................................................... 12
    Tuning the Oracle BPEL Service Engine ............................................... 13
      Tuning BPEL Thread Pools .................................................................... 14
      Tuning Timeouts and Audit Level .......................................................... 14
    Tuning the Oracle Mediator Service Engine ........................................... 15
  Oracle AIA ....................................................................................................... 16
Monitoring Guidelines ..................................................................................... 17
Test Configuration ............................................................................................. 19
  Hardware ........................................................................................................ 19
  Software ........................................................................................................ 19
  Topology ........................................................................................................ 19
Performance Tuning of Asynchronous Flows .............................................. 22
  Oracle AIA Communications O2C Pre-built Integration Flow ............... 22
  Test Simulation ............................................................................................... 24
Baseline Test........................................................................25
Adapter Tuning........................................................................27
  Observation.........................................................................27
  Analysis.............................................................................27
  Implementation.................................................................28
  Results.............................................................................28
Resequencer Tuning ..............................................................30
  Observation.........................................................................30
  Analysis.............................................................................30
  Implementation.................................................................30
  Results.............................................................................31
Adapter Tuning (Increase Threads on all Consumers)............32
  Observation.........................................................................32
  Analysis.............................................................................32
  Implementation.................................................................34
  Results.............................................................................34
Increase Order Rate ..............................................................36
  Observation.........................................................................36
  Analysis.............................................................................36
  Implementation.................................................................36
  Results.............................................................................36
Memory Tuning.......................................................................37
  Observation.........................................................................37
  Analysis.............................................................................37
Implementation........................................................................... 39
Results ......................................................................................... 40
Vertical Scaling of SOA Nodes ................................................. 42
  Observation ............................................................................. 42
  Analysis ................................................................................ 42
  Implementation ..................................................................... 42
  Results .................................................................................. 43
Adapter Tuning (Decrease threads) ......................................... 45
  Observation ............................................................................. 45
  Analysis ................................................................................ 45
  Implementation ..................................................................... 45
  Results .................................................................................. 46
Database Tuning ........................................................................ 49
  Observation ............................................................................. 49
  Analysis ................................................................................ 49
  Implementation ..................................................................... 50
  Results .................................................................................. 50
Audit Tuning ............................................................................. 52
  Observation ............................................................................. 52
  Analysis ................................................................................ 52
  Implementation ..................................................................... 53
  Result ..................................................................................... 53
Response Time Analysis .............................................................. 54
Summary ..................................................................................... 55
Performance Tuning of Synchronous Flows ............58
  Oracle AIA AABC Pre-built Integration Flow ..................58
  Test Simulation..................................................58
  Test Results.....................................................59
Conclusion..........................................................60
Appendix A: SQL Queries used for Monitoring ........61
Appendix B: Tuning Oracle JRockit..........................63
Introduction

Oracle’s AIA Foundation Pack (FP) and pre-built integrations provide an accelerated approach for integrating enterprise applications. Just like any other integration in an enterprise, an AIA FP integration or an AIA pre-built integration must be tuned to handle an enterprise’s message volumes, performance requirements, and Service Level Agreements (SLAs). This document describes the guidelines and results for the performance tuning of AIA 11.x.

The document is organized into six chapters. The Guidelines chapter provides a step by step approach to tuning AIA deployments. It should be noted that the guidelines suggested are just one approach to tuning AIA. They also provide the list of the most important tuning knobs available to tune AIA. Since AIA is built on top of Oracle FMW, the same tuning guidelines can be applied for tuning custom integrations developed using Oracle FMW. Oracle FMW is a large product suite and this document focuses on tuning particular components such as BPEL, Mediator and Adapters. Apart from tuning Oracle FMW components, this whitepaper covers all the relevant areas of tuning that would apply to an AIA deployment including infrastructure database tuning, application server (Weblogic) tuning and Java Virtual Machine (JVM) tuning.

The Test Configuration chapter describes the hardware, software and topology used to perform the performance tuning exercise. The performance tuning exercise was done on Oracle’s SPARC T4 servers running Oracle Solaris 10.

The Performance Tuning of Asynchronous Flow and Performance Tuning of Synchronous Flow chapters showcase performance tuning for both asynchronous and synchronous integration flows. The performance tuning exercises for an asynchronous flow were done on an out-of-the-box Communications Order to Cash (O2C) Integration Pack for Siebel CRM, Oracle Order and Service Management, and Oracle Billing and Revenue Management. The Communications O2C pre-built integration implements a complex integration with several endpoint interactions, decoupling and so on and therefore is a good choice for this exercise. The synchronous flow adopted the tuning that was mainly done as a part of asynchronous
flow. Communications Agent Assisted Billing Care (AABC) pre-built integration was used for the synchronous flow. It should be noted that most of the tuning activities found in this guide apply both to synchronous and asynchronous integration flows though they may appear in only one context.

The Conclusion chapter summarizes the performance tuning results.

While you may pick certain points of interest from this white paper, we recommend that you read it from end-to-end to get the full context. The document is organized as a flow, from generic to specific.
Guidelines

Performance Testing Guidelines

This section introduces general guidelines to conduct performance testing on Oracle AIA pre-built integrations. The methodology involves starting with the pre-built integration as delivered and tuning various layers until the maximum performance is reached. Here, the term 'maximum performance' should be read in the context of the desired performance goal.

**GOAL:**

- Achieve maximum order throughput (transactions per minute)
- Achieve the best response time

**NOTE:** The Transactions Per Minute (TPM) in this white paper refer to the number of orders flowing through AIA per minute. This should not be confused with the TPM definition on WLS or the database.

The two goals listed above may act in a contradictory way. When maximum TPM are achieved, all transactions may not have the best response time. This is explained in detail in later sections. For the specific performance tuning exercise discussed in this document the goal is to maximize the order throughput while observing the response time variances.
Guidelines: Diagram 1

**Baseline**

Baselining involves determining the initial value for the number of transactions and the delay between the transactions. For this exercise the Number of Transactions (Orders) = 500 and the Delay between Orders = 5 seconds were chosen as the baseline values. We recommend that you do not start tuning with peak level values so choose baseline values that are lower than the peak level values. For example if “X” is the expected peak level load, then the baseline could be set at “X/10”.

**Run Tests**

The tests during the performance tuning were simulated using SoapUI. SoapUI enqueued a batch of new orders into the same queue that Siebel would have used to place orders to be picked up by the pre-built integration. SoapUI was configured to pump in orders with unique order numbers. Medium sized orders (30 order lines items per order) were used in the current exercise. The methodology does not restrict the choice of tool to be used for load generation. SoapUI was chosen for the current tuning exercise due to its “fit for purpose” and flexibility.
After every server bounce you should warm the system with test orders before running the actual test. A “warm up” of the environment ahead of starting the actual test allows the environment to reach a “steady-state” condition.

To test run in a real life scenario, most of the test runs were made without purging the data from the dehydration database. However, there were tests that were run after the db was purged to record the performance benefits.

**Tune the System**

The tuning of pre-built integration system can be performed at several levels. As shown in the diagram, AIA is the top most layer; however, in order to achieve maximum performance, all the layers in the stack must be tuned appropriately.

![Test System Layers: Diagram 1](image)

**Increase Transaction Rate**

Although the performance tests are started with significant delay between the transactions, the ultimate goal is to get the maximum TPM. To achieve maximum TPM, the order injection rate of the load generation tool is incrementally increased as the system is progressively tuned. The count of transactions must be increased and the delay between transactions must be reduced until there is no delay at all. A zero delay simulates all the transactions coming in at the same time.
Tuning Knobs Guidelines

This section provides general guidelines and best practices for tuning the components in the various layers.

Operating System

Operating system tuning should set configurations and parameters according to the install guides. Because of the variety of operating systems and versions that can be run, refer to the latest release notes and [https://support.oracle.com](https://support.oracle.com) for configuring and tuning the OS.

High performance architecture separates the application tier from the database tier. In part this allows each tier to be configured and tuned for the different requirements of the application servers and the database servers.

JVM

This section covers the tuning guidelines for the Java Virtual Machine (JVM).

Note that JVM tuning guidelines discussed in this section are based on the following:

- JDK 1.6.x (Hotspot and JRockit)
- Modern server class boxes (64-bit, 4+ cores, 16GB+ RAM)

In general, Hotspot JVM is recommended on Oracle SPARC servers running Oracle Solaris. The following section discusses the Hotspot JVM tuning.

For JRockit-specific tuning guidelines, refer to “Appendix B: Tuning Oracle JRockit“.

Tuning Hotspot JVM

The Hotspot JVM is the reference JVM implementation which uses a generational heap. A generational heap separates the storage area of young generation and tenured objects. Objects are promoted from the young generation to the old generation after they have survived a configurable number of garbage collection cycles.

Tuning the Memory Footprint
The following diagram shows all the parameters required to size the heap and generations.

JVM: Diagram 2

When a Hotspot JVM starts, the heap, the young generation and the perm generation space are allocated to their initial sizes determined by the -Xms, -XX:NewSize, and -XX:PermSize parameters respectively, and increment as-needed to the maximum reserved size, which are -Xmx, -XX:MaxNewSize, and -XX:MaxPermSize. The JVM may also shrink the real size at runtime if the memory is not needed as much as originally specified. However, each resizing activity triggers a Full Garbage Collection (GC), and therefore impacts performance. As a best practice, we recommend that you make the initial and maximum sizes identical.

The general guidelines for tuning the JVM memory are as follows:

- Use larger heaps whenever possible on modern servers (64 bit): For example, use 12G or 16G as the default configuration for -Xmx, and then resize it as needed.
- Set the initial size equal to the maximum size:
  - -Xms = -Xmx
  - -XX:NewSize = -XX:MaxNewSize: Or simply set –Xmn
  - -XX:PermSize = -XX:MaxPermSize: Set it to 512m will suffice in most AIA scenarios.

Enabling GC Logs

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>GUIDELINES</th>
</tr>
</thead>
<tbody>
<tr>
<td>-verbose:gc</td>
<td>Enabling GC logs is essential in performance monitoring and GC tuning as it can</td>
</tr>
</tbody>
</table>
showcase the GC behavior and statistics, e.g. how often does one GC (minor or full GC) happen, and how long does each GC take, etc. GC logs are also useful for determining the possible memory leaks. For example, a GC analyzing tool can be used to analyze the GC log to figure out the message consumption trend. For example GCViewer from github

Note:
<location> is the GC log file name including the absolute path

Tuning Garbage Collections
Since JVMs use GC policies to determine the algorithm for garbage collection, choosing the right policy is the key for tuning GC performance. This section provides guidelines for using the suggested GC policy settings.

- The following settings maximize overall throughput at the expense of high garbage collection pauses overhead:

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>DEFAULT VALUE</th>
<th>GUIDELINES</th>
</tr>
</thead>
<tbody>
<tr>
<td>-XX:+UseParallelGC</td>
<td>Default in JDK 6</td>
<td>This policy uses an algorithm that collects the young generation using multiple concurrent threads. Since collecting young generation is a stop-the-world phase of garbage collection, performance is significantly improved by taking advantage of multiple CPU cores. However, this policy uses a single thread approach when collecting the old generation.</td>
</tr>
<tr>
<td>-XX:+UseParallelOldGC</td>
<td>ParallelOld is the new algorithm introduced in JDK 6 to collect the old generation using multiple threads. This is the recommended setting for the AIA environment in which high throughput is preferred over low latency.</td>
<td></td>
</tr>
</tbody>
</table>

- The following setting can be used to achieve low latency:

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>DEFAULT VALUE</th>
<th>GUIDELINES</th>
</tr>
</thead>
<tbody>
<tr>
<td>-XX:+UseConcMarkSweepGC</td>
<td>This policy uses the Concurrent Mark Sweep (CMS) algorithm for managing the old generation. This algorithm is mostly concurrent, and can eliminate or reduce the frequency of full stop-the-world collections.</td>
<td></td>
</tr>
</tbody>
</table>

- Use the following setting to control the number of GC threads:

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>DEFAULT VALUE</th>
<th>GUIDELINES</th>
</tr>
</thead>
<tbody>
<tr>
<td>-XX:ParallelGCThreads</td>
<td>If not specified, JVM will determine the number of GC threads based on the CPU cores of the hardware. For high end servers that have multiple CPU cores, it is the best practice to limit the number of GC threads by setting this parameter to an appropriate value. For example, for a machine that has 64 cores, this parameter can be set to 16 or 32. Therefore other JVMs on that machine can still have CPU resource when GC happens.</td>
<td></td>
</tr>
</tbody>
</table>
## Tuning Other Parameters

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>DEFAULT VALUE</th>
<th>GUIDELINES</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-XX:+HeapDumpOnOutOfMemoryError</code></td>
<td></td>
<td>With this setting, the heap dump is generated when an Out of Memory error occurs.</td>
</tr>
<tr>
<td><code>-XX:HeapDumpPath=&lt;location&gt;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>-XX:+UseCompressedOops</code></td>
<td></td>
<td>This flag:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Enables the use of compressed 32 bit Ordinary Object Pointers in 64 bit JVM.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Reduces overall heap usage and also increases CPU cache efficiency.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Is enabled by default in Hotspot JVM since version 1.6u23</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note when using large heaps that the pointer compression feature is lost for heap sizes of 32GB and above.</td>
</tr>
<tr>
<td><code>-XX:SurvivorRatio</code></td>
<td>8</td>
<td>This is the ratio of Eden space to survivor space sizes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Used to tune the size of the survivor spaces. A smaller ratio provides bigger survivor spaces and can be used to delay the tenuring of objects.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Used for fine tuning GCs based on analysis of GC logs.</td>
</tr>
<tr>
<td><code>-XX:TargetSurvivorRatio</code></td>
<td>50</td>
<td>This defines the desired percentage of survivor space to be used after scavenging.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Increasing this can yield a higher tenuring threshold and delays the promotion of objects to the tenured space. Used for fine tuning GCs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>based on analysis of GC logs.</td>
</tr>
</tbody>
</table>

## Database

SOA/AIA components use the database extensively for product functions such as message delivery, dehydration, cross references, and required database operations. Thus, tuning the underlying database is essential to achieve optimized performance and we recommend that you restrict database persistence to the minimum level required. Detailed database tuning is often dependent on the database server hardware and the application and thus is beyond the scope of this paper.

There are some generic guidelines for tuning database:

- Set basic parameters correctly. Basic parameters include processes, memory settings, table spaces, etc.

- Create redo logs to a size so that they switch only every 20 minutes. The size from a default database installation is 5 MB. In initial warm-up tests these logs were switching every 20 seconds. One gigabyte logs are a good starting point, and increasing from there may be necessary.
• Run the AWR report along with necessary instrumentation on other layers (OS, Weblogic server, SOA and AIA). This can help narrow down the performance bottleneck. If a problem is found in the database layer, tune as needed.

• Look for the top SQL in the AWR report. Review the SQL which is performing poorly and tune.

This table shows a starting point for database initialization parameters.

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>DEFAULT VALUE</th>
<th>GUIDELINES</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISTRIBUTED_LOCK_TIMEOUT</td>
<td>60</td>
<td>This parameter specifies the amount of time (in seconds) for distributed transactions to wait for locked resources.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DISTRIBUTED_LOCK_TIMEOUT &gt;= JTA Timeout</td>
</tr>
<tr>
<td>db_securefile</td>
<td>PERMITTED</td>
<td>Set to “ALWAYS” before creating schema. This creates all LOBs as SecureFile LOBs that are in an Automatic Segment Space Managed tablespace.</td>
</tr>
<tr>
<td>sga_target</td>
<td></td>
<td>These are some of the main parameters which control the memory used by the database.</td>
</tr>
<tr>
<td>pga_aggregate_target</td>
<td></td>
<td>There are many more which can fine tune use of memory but are beyond the scope of this document.</td>
</tr>
<tr>
<td>memory_target</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

There are some key performance indicators in the AWR report that show if an AIA system is healthy and performing well:

• Top 5 timed foreground events: This should normally show “DB CPU” and “db file sequential read” as top wait class/event. Every other wait class on top level needs to be investigated.

• SQL Statistics – ordered by elapsed time: All SQL statements for AIA normally should execute far below 1 second (on a tuned system typically in less than 50ms). Check especially the execution of queries on XREF_DATA in the AIA context.

• SGA Target Advisory: This indicates that the size of the SGA is too small.

• PGA Memory Advisory: This indicates that the size of the PGA is too small.
- Segments by Row Lock Waits, Segments by ITL Waits, Segments by Buffer: These sections give more information about objects causing high wait times on DB side.

- Segments by Physical Reads: This report section, in conjunction with other AWR report sections, identifies the primary cause of really hot segments.

- Wait Event Statistics - Wait Class: Normally “CPU time”, “Application” and “User I/O” are the top wait classes. If, for example, “Cluster” shows high average wait times, you need to check the RAC setup.

- Wait Event Statistics - Wait Events: Typical indicators when performance is not optimal with AIA show high waits/events of “enq: TX - row lock contention” or “enq: TX – contention”. If wait class “enq: HW – contention” shows high values, use secure files for LOB tables.

This table shows a poorly performing database- indicated by the “Top 5 Timed Foreground Events”:

<table>
<thead>
<tr>
<th>Event</th>
<th>Waits</th>
<th>Time(s)</th>
<th>Avg Wait(ms)</th>
<th>% Total Call Time</th>
<th>Wait Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>enq: TH - contention</td>
<td>81,057</td>
<td>40.148</td>
<td>495</td>
<td>75.9</td>
<td>Application</td>
</tr>
<tr>
<td>enq: TX - row lock contention</td>
<td>15,532</td>
<td>6,923</td>
<td>446</td>
<td>13.1</td>
<td>Application</td>
</tr>
<tr>
<td>CPU time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Application</td>
</tr>
<tr>
<td>db Fav sequential read</td>
<td>95,768</td>
<td>755</td>
<td>8</td>
<td>1.4</td>
<td>User I/O</td>
</tr>
<tr>
<td>SQL*Net break/reset to client</td>
<td>54,668</td>
<td>318</td>
<td>6</td>
<td></td>
<td>Application</td>
</tr>
</tbody>
</table>

**Tuning Oracle Weblogic Server**

This section focuses on AIA-related tuning for Weblogic Server. The general Weblogic Server tuning topics are beyond the scope of this white paper.

Refer to *Performance and Tuning for Oracle WebLogic Server* for more information.

**JTA Transaction Timeout**

The Java Transaction API (JTA) transaction timeout is a global server level configuration, which means tuning this parameter may affect all applications targeted to this server. In a typical SOA/AIA environment, one or more Weblogic managed servers act as a dedicated server for AIA related applications, so you should keep this setting the same as the SOA Enterprise JavaBeans (EJB) transaction timeout (300 seconds by default). Since this parameter is not used to speed up a transaction you should tune it only if particular transactions need a longer time to complete.
JDBC Connection Pools

The Java Database Connectivity (JDBC) connections are extensively used by the SOA server. To ensure there are enough connections available during runtime you should tune the following data sources:

- mds-soa
- oraSDPMDatasource
- SOADatasource
- SOALocalTxDatasource
- XrefDatasource

The guidelines for tuning these data sources and connection pools are listed below:

- The default values for “Initial Capacity”, Maximum Capacity” and “Minimal Capacity” are low. Increase them before load testing.
- During runtime, monitor the JDBC matrix, especially Waiting For Connection Failure Total, Wait Seconds High Count and Failed Reserve Request Count, which will provide insight into the performance of the JDBC data sources. These parameters can be monitored from the Weblogic console.

More information on JDBC monitoring parameters can be found in the documentation at http://docs.oracle.com/cd/E23943_01/apirefs.1111/e13952/pagehelp/Corecoreserverservermonitoringjdbctitle.html.

Tuning Oracle SOA Infrastructure

Oracle SOA Infrastructure is the key contributor to the overall performance of AIA. This section covers the guidelines and common best practices for tuning Oracle SOA Infrastructure. We recommend that you perform the following tuning tasks:

- Tune Oracle Java Connector Architecture (JCA) Adapters, such as JMS Adapter
- Tune the Oracle BPEL Service Engine
- Tune the Oracle Mediator Service Engine

Tuning Oracle JMS Adapters

The JMS queue-based integration is commonly used to integrate various systems asynchronously in an AIA environment. In a typical AIA scenario, such as Order to Cash, the JMS Adapter is configured in a SOA composite to consume messages from a JMS queue.

The following table lists the most important parameters for tuning the JMS adapter.

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>DEFAULT VALUE</th>
<th>GUIDELINES</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


adapter.jms.receive.threads

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Set the initial value to a lower number, e.g., 4. In many cases a JMS Adapter simply consumes and then persists messages into a database instance. Setting the value to a higher number, e.g. 20 or more, may help improve the overall performance. However, in an AIA scenario, it is not the case and is often the opposite; which means the higher the number, the worse the performance. This is caused by more parallel transactions and resource contention. In most AIA asynchronous integration patterns, the JMS poller threads are also the working threads between milestones and execute the SOA components (BPEL, Mediator, SOAP calls, etc.). A high number of working threads may cause high context switching, and consume more system resources (CPU, Memory, I/O, etc.). If there are only one or two queues, it should not be a problem, however if the system has multiple queues (more than 6 or 7) the total number of threads will increase dramatically, thus impacting performance. Do not set this parameter larger than 20. This parameter can be increased from the initial value to up to 20 if there are more resources available in the system. When tuning this parameter in a High Availability (HA) environment where the JMS poller threads listen on the distributed queues, the total number of actual listening threads at runtime on each queue is adapter.jms.receive.threads multiplied by the number of nodes in the cluster. So keeping this parameter to a lower value is always the best practice. Keep monitoring all the JMS queues and ensure there is no congestion in one or more queues. If a congestion situation is detected, fine tune this parameter for related queues.</td>
</tr>
</tbody>
</table>

adapter.jms.receive.timeout

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 second</td>
<td>This parameter specifies the time value for the receive API call. The default value may not be enough when processing a large payload.</td>
</tr>
</tbody>
</table>

Max Capacity for the Outbound Connection Pool

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>The property specifies the max capacity for the outbound connection pool of the JMS Adapter. Increase the value under heavy load. Otherwise, the following error messages will be seen in the log file: Unable to get a connection for pool = &quot;&quot;, weblogic.common.resourcepool.ResourceLimitException: Configured maximum limit of (0) on number of threads allowed to wait for a resource reached for pool &quot;&quot; at weblogic.connector.outbound.ConnectionManagerImpl.getConnectionInfo(ConnectionManagerImpl.java:422)</td>
</tr>
<tr>
<td></td>
<td>This property is managed by the Weblogic Server and can be modified through the Weblogic Server Administration Console.</td>
</tr>
</tbody>
</table>

**Tuning the Oracle BPEL Service Engine**

The BPEL Service Engine provides key functionalities, such as message delivery, persistence, process execution, audit trail, and service invocation, in a typical SOA/AIA scenario. Having a highly optimized BPEL service engine should be a major objective of any AIA performance tuning.

The following topics are covered in this section:

- Tuning BPEL thread pools
- Tuning timeouts and Audit Level
**Tuning BPEL Thread Pools**

When a BPEL Service Engine starts up, it creates three thread pools as follows:

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>DEFAULT VALUE</th>
<th>GUIDELINES</th>
</tr>
</thead>
</table>
| Dispatcher Engine Pool| 30            | Threads in this pool are created to process engine dispatcher messages. Engine dispatcher messages are generated whenever an activity (e.g., callback) must be processed asynchronously. The default value is 30 threads, which requires tuning in the AIA context. The guidelines for tuning the Engine Threads Pool are as follows:  
  - Monitor the Engine statistics during load testing, and use the threads dump to help analyze engine threads behavior.  
  - Gradually increase the engine threads pool to see if any performance gains are achieved.  
  - Do not over size the engine pool (should be less than 200 in most cases). Higher threads can cause more resource consumption and may impact performance.  
  Changing the threads pool size requires you restart the SOA server. |
| Dispatcher Invoke Pool| 20            | Threads in this pool are used to process invocation dispatcher messages which are meant to instantiate a new instance. The default value is 20 threads. This pool is not being used in many AIA implementations, like Communication O2C pre-built integration, because JMS Adapter threads are usually the ones that instantiate BPEL processes. |
| Dispatcher System Pool| 2             | Threads are allocated to process system dispatcher messages. The default value is 2 threads. The default value should suffice in most cases. Tuning is not required. |

**Tuning Timeouts and Audit Level**

Transaction timeout and rollback related errors are common errors that occur under a heavy load situation indicating either the transaction timeout properties are not properly configured, or a potential performance issue exists in one of the systems. We recommend that you set the timeout property to a reasonable value (such as 300 seconds), and do not increase it to a value higher than 600 unless there is a long-running transaction required by the business. Basically, setting the higher value only hides the problem, and does not solve it.

Tune the following timeout related settings as required:

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>DEFAULT VALUE</th>
<th>GUIDELINES</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPEL EJB timeout</td>
<td>300 Seconds</td>
<td>This is the EJB level timeout that forces the BPEL engine components (e.g., engine beans, delivery beans, etc.) to time out after a certain amount of time. The default value is 300 seconds, which should suffice in most cases.</td>
</tr>
<tr>
<td>Sync Max Wait Timeout</td>
<td>45 Seconds</td>
<td>This attribute specifies the time that the invoker threads will wait for a response to return from a synchronous durable process. Even though it is only applicable for synchronous durable processes, this parameter is often set to a higher value along with other transaction timeout properties as a proactive approach.</td>
</tr>
<tr>
<td>Audit Level</td>
<td>Production</td>
<td>Tuning parameter Audit Level is set to “production” by default. Turning it off can make significant performance improvement, however the trade-off is that the auditing capability is</td>
</tr>
</tbody>
</table>
lost. If performance is the priority, set this property to "off" in the composite level, and then set it to "production" in the BPEL or Mediator level.

| auditStorePolicy     | syncSingleWrite | syncMultipleWrite - stores audit data synchronously when it saves the cube instance. This is done as part of the cube instance transaction.  
SyncMultipleWrite - stores audit data synchronously using a separate local transaction. Synchronously means the BPEL service engine is saving the audit trail in the same thread as the main transaction. However, it is doing it in a "separate local transaction".  
Because they are on the same thread, latency of saving the audit trail is still included into the overall latency of the main transaction. However, because they are on separate transactions, the BPEL Engine can be configured (using AuditFlushByteThreshold and AuditFlushEventThreshold) to flush out the audit trail from memory to the database periodically, regardless of how long the main transaction takes. Moreover, having them on two separate transactions means the rollback of the main transaction does NOT affect the audit trail transaction. That is, an audit trail will be seen even if the main transaction rolls back.  
async - stores the audit data asynchronously using an in-memory queue and pool of audit threads. The advantage is the audit trail latency is NOT directly included in the latency of the main transaction (but because they still share the computing resources and database, the latency is still indirectly related). The disadvantage is that because audit trails are being saved asynchronously, the audit trail may be out of sync from the main transaction (as the name 'async' implies).  
AuditFlushByteThreshold | 2 MB | When auditStorePolicy=syncMultipleWrite or auditStorePolicy=async, these two flags control how often the engine should flush the audit events. These two properties do NOT apply to auditStorePolicy=syncSingleWrite. auditFlushByteThreshold means that after adding an event to the current batch, the engine checks if current batch byte size is greater than this value or not. If "yes", then it flushes the current batch. The default value is 2048000 (bytes), i.e. 2MB. Similarly, auditFlushEventThreshold means that when this limit is reached, the engine triggers the store call. The default value is 300 (events).  
AuditFlushEventThreshold | 300 | Tuning the Oracle Mediator Service Engine  
The Mediator Service engine itself does not need much tuning unless the resequencer is implemented in one or more Mediator service components. Some Oracle AIA pre-built integrations, such as Communications O2C, require setting up a resequencer, which changes a synchronous routing service to an asynchronous one. The default settings of the resequencer are not optimized for system performance and should be tuned accordingly. The following table lists the parameters for tuning the resequencer.

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>DEFAULT VALUE</th>
<th>GUIDELINES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resequencer Locker</td>
<td>10 seconds</td>
<td>This parameter specifies the amount of time in seconds for the resequencer locker thread to sleep between iterations of locking. By default, the single locker thread wakes every 10 seconds to start locking groups, which significantly impacts the throughput in a heavy load.</td>
</tr>
<tr>
<td>Thread Sleep(sec)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
situation. The best practice is to decrease this value on a production environment. For instance, it can be set to 1 (second) in many cases.

<table>
<thead>
<tr>
<th>Resequencer Maximum Groups Locked</th>
<th>4</th>
<th>This parameter defines the maximum number of groups to be locked by the locker thread in a single iteration. By default, the locker threads can only lock up to 4 groups once a time so this should be increased to improve the throughput under heavy load. Monitoring the resequencer tables sheds light on the need for tuning this parameter. Refer to &quot;Appendix A: SQL Queries used for Monitoring&quot; for the sql queries used for resequencer monitoring.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resequencer Worker Threads</td>
<td>4</td>
<td>This parameter defines the number of worker threads that process the locked groups. The default value is 4, which should be increased in most cases. The guideline for tuning this parameter is to set an initial value considering the settings of the BPEL engine threads and the Adapter poller threads, and then tune this parameter whenever the resequencer becomes a bottleneck. Note that you must bounce the server bounce after the change.</td>
</tr>
</tbody>
</table>

### Oracle AIA

The common practice for tuning AIA composites is to tune the dehydration options for synchronous BPEL processes. By default, the state information gets dehydrated after the completion of a synchronous process. To improve performance, set the following properties in the composite.xml file.

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>DEFAULT VALUE</th>
<th>GUIDELINES</th>
</tr>
</thead>
<tbody>
<tr>
<td>bpel.config.inMemoryOptimization</td>
<td>false</td>
<td>Set to “true” for synchronous BPEL processes.</td>
</tr>
<tr>
<td>bpel.config.completionPersistencePolicy</td>
<td>all</td>
<td>Set to “faulted” for synchronous BPEL processes to dehydrate only faulted instances.</td>
</tr>
</tbody>
</table>
Monitoring Guidelines

It is essential to constantly monitor the health and performance of the system for the tests performed. This table lists the monitoring parameters and provides general guidelines and examples of how the monitoring was implemented for Communications O2C.

<table>
<thead>
<tr>
<th>MONITORING PARAMETER</th>
<th>GUIDELINES</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Utilization</td>
<td>Monitor the vital statistics of the CPU to gauge utilization and identify opportunities for scale-out. Used the ‘sar’ Oracle Solaris utility command to log the CPU Utilization during the test execution.</td>
</tr>
<tr>
<td>Disk Usage Statistics</td>
<td>Monitor the disk usage to identify read/write bottlenecks in file persistent stores under load. Used the ‘iostat’ utility command to log the I/O statistics during the test execution.</td>
</tr>
<tr>
<td>BPEL threadpools</td>
<td>Monitor the load on the BPEL threadpools to ascertain the performance and identify bottlenecks. Used JMX Mbeans to measure values for Invoke, Engine, System and Audit threadpools. Refer to <a href="http://stefankoser.blogspot.com/">http://stefankoser.blogspot.com/</a> for details about the JMX utility used for thread pool monitoring.</td>
</tr>
<tr>
<td>AIA Queues</td>
<td>Monitor the load and message depth on the Integration JMS Queues to identify bottlenecks and impact of tuning. Used JMX Mbeans to capture Current Messages Count values of the following Communications O2C JMS Queues. AIA_SALESORDERJMSQUEUE, AIA_CRTFO_IN_JMSQ, AIA_CRTCUST_OUT_JMSQ, AIA_CRTBO_OUT_JMSQ, AIA_UPDCUST_IN_JMSQ, AIA_UPDBO_IN_JMSQ, AIA_UPDSO_OUT_JMSQ. Refer to <a href="http://stefankoser.blogspot.com/">http://stefankoser.blogspot.com/</a> for details about the JMX utility used for AIA Queues monitoring.</td>
</tr>
<tr>
<td>Resequencer</td>
<td>Monitor the resequencer tables for build up messages, and to identify bottlenecks and the impact resequencer parameter tuning. Used DB scripts to query resequencer tables mediator_group_status and mediator_resequencer_message and periodically logs message count for the component_dns of interest. For more information about the sql queries used for monitoring, refer to “Appendix A: SQL Queries used for Monitoring”.</td>
</tr>
<tr>
<td>Garbage Collection Statistics</td>
<td>Monitor the GC activity to identify heap usage under load and identify GC tuning areas. Standard JVM Garbage collection logs obtained by the following Hotspot JVM parameters -XX:+PrintGCDateStamps -XX:+PrintGCDetails --Xloggc:/perfaia/monitors/WLS_SOA2.gc.hotspot.log -verbose:gc</td>
</tr>
</tbody>
</table>
For equivalent JRockit parameters see ‘Appendix B: Tuning Oracle JRockit’.

| Performance Goals: Order Throughput and Avg Response Time | Monitor the performance goals and other vital statistics of the system under test. The data and observation guides the direction for the performance tuning. The throughput, average response time, resequencer locker and worker thread delays etc were monitored regularly by querying the backend DB. For more information about the sql queries used for monitoring, see “Appendix A: SQL Queries used for Monitoring”. |
| Database Statistics | Monitor the health and performance of the database to identify DB bottlenecks that may require DB tuning. The DB health and performance was constantly monitored using the DB EM console, alert logs, and AWR reports. |
| SOA Server Health | Monitor the server logs to identify exceptions, their cause during the performance tuning. Some of the tuning performed may lead to adverse side effects. SOA Server health was constantly monitored using the SOA EM console and server diagnostic logs. |
| JDBC Connection Pools | Monitor the JDBC Connection Pools for analyzing resource bottlenecks and opportunities to tune JDBC pools. The JDBC Connection pools were monitored from the WLS console. Server->Monitoring->JDBC. |
| JCA Adapter Connection Pools | Monitor the JCA Adapter Connection Pools for analyzing resource bottlenecks and opportunities for tuning. The JDBC Connection pools were monitored from the WLS console. Adapter->Monitoring->Outbound Connection Pools. |
Test Configuration

Hardware

The test described in this document was done on Oracle Solaris, Oracle SPARC T4 servers, and Oracle’s Sun Storage 2540-M2 Fiber Channel array. The test environment was configured to be able to drive the maximum amount of load in order to characterize performance and scalability.

The sections below provide more detailed information on both the hardware and software setup along with guidance on how to most effectively utilize the system resources.

All servers reside in the same subnet, with dedicated 1/10 Gigabit Ethernet as appropriate. The table below lists the specifications for each system.

<table>
<thead>
<tr>
<th>MODEL</th>
<th>USE</th>
<th>NUMBER OF SERVERS</th>
<th>CPUS</th>
<th>CORES PER CPU</th>
<th>CPU STRANDS PER CORE</th>
<th>TOTAL HW STRANDS</th>
<th>TOTAL MEMORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPARC T4-2</td>
<td>SOA/AIA Server</td>
<td>1</td>
<td>2x2.85 GHz SPARC T4</td>
<td>8</td>
<td>8</td>
<td>128</td>
<td>128GB</td>
</tr>
<tr>
<td>SPARC T4-2</td>
<td>Database</td>
<td>2</td>
<td>2 x 2.85 GHz SPARC T4</td>
<td>8</td>
<td>8</td>
<td>128</td>
<td>128GB</td>
</tr>
</tbody>
</table>

Software

The table below lists the software and versions used for the various layers of the AIA Integration.

<table>
<thead>
<tr>
<th>COMPONENT</th>
<th>VERSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIA Communications PIPs</td>
<td>11.2</td>
</tr>
<tr>
<td>Oracle SOA Suite</td>
<td>11.1.1.6 (PS5)</td>
</tr>
<tr>
<td>Oracle Weblogic</td>
<td>10.3.6</td>
</tr>
<tr>
<td>Hotspot</td>
<td>JDK 1.6 (1.6.0_33-b03)</td>
</tr>
<tr>
<td>Oracle Database Server</td>
<td>11.2.0.2</td>
</tr>
<tr>
<td>Oracle Solaris</td>
<td>Oracle Solaris 10 update 10</td>
</tr>
</tbody>
</table>

Note: The Hotspot JVM was used for this performance exercise. We recommend that you use Hotspot JVM for Solaris SPARC platform.

Topology

The initial topology chosen for the performance tuning exercise consisted of a 2 node App cluster on 2 SPARC T4 zones residing on 1 physical box. The RAC database resided on 2 SPARC T4 physical machines. Note that this was just a chosen topology. Refer to the ‘Enterprise Deployment Guide’ for detailed discussion on topologies. This diagram illustrates the topology:
The exercise was later scaled out to a 4 node cluster spread across 2 zones of a physical SPARC T4 machine. The RAC database resided on 2 SPARC T4 physical machines. The topology is as shown in the diagram:
Topology: Diagram 2
Performance Tuning of Asynchronous Flows

This section covers the tests done during the tuning exercise. The AIA Communications O2C asynchronous flow was chosen for this Performance Tuning. The aim of the exercise is to tune the AIA, SOA, Weblogic, JVM and OS layers to arrive at the best throughput for the given Communications O2C PIP.

Oracle AIA Communications O2C Pre-built Integration Flow

AIA Communications Order to Cash (O2C) leverages the queue-based integration pattern to integrate Siebel Customer Relationship Management (Siebel CRM), Oracle Order and Service Management (OSM), and Oracle Billing and Revenue Management (Oracle BRM). The following diagram depicts a typical message flow in this solution.

O2C Flow Diagram 1

This complex end-to-end message flow can be further decomposed into the following sub-flows:
When a sales order is captured in the Siebel CRM system for processing, Siebel CRM sends the order to the AIA_SALESORDERJMSQUEUE JMS queue, which is the starting point for the Communications O2C flow.

The message is consumed by ProcessSalesOrderFulfillmentSiebelCommsJMSConsumer through a JMS Adapter, and then passed to ProcessSalesOrderFulfillmentSiebelCommsReqABCSImpl, which transforms the payload to Enterprise Business Message (EBM) and sends it to the AIA_CRTFO_IN_JMSQ queue.

OrderOrchestrationConsumer consumes the message and then sends it to TestOrderOrchestrationEBF, which is a business flow simulating an order orchestration service. At the end of transaction, a Create Customer business message is sent to AIA_CRTCUST_OUT_JMSQ.

ProcessFulfillmentOrderBillingAccountListOSMCFScommsJMSConsumer consumes the message and triggers this subflow. The key composite in this flow is CommsProcessBillingAccountListEBF, which first queries the customer details through QueryCustomerPartyListSiebelProvABCSImplV2, and then synchronizes the customer details to BRM. In the end, the sync customer result is sent back to the AIA_UPDCUST_IN_JMSQ. Note that the flow spans multiple transactions as a result of the asynchronous BPEL process and the resequencer.

CustomerResponseConsumer consumes the message from AIA_UPDCUST_IN_JMSQ, and then invokes TestOrderOrchestrationEBF, which sends a message to AIA_CRTBO_OUT_JMSQ to trigger the sub-flow for business object processing in Oracle BRM.

ProcessFulfillmentOrderBillingOSMCFScommsJMSConsumer starts the flow by consuming the message from the AIA_CRTBO_OUT_JMSQ queue. After that, the Application Business Connector Service (ABCS) composite ProcessFulfillmentOrderBillingBRMCommsProvABCSImpl communicates with BRM and the result is sent back to AIA_UPDBO_IN_JMSQ.

TestOrderOrchestrationEBF is triggered again after a message arrives in the AIA_UPDBO_IN_JMSQ queue, and places the updated order payload in AIA_UPDSO_OUT_JMSQ.

UpdateSalesOrderOSMCFScommsJMSConsumer consumes the message from AIA_UPDSO_OUT_JMSQ, and then invokes the UpdateSalesOrderSiebelCommsProvABCSImpl composite, which sends the updated order to Siebel CRM. This is the end of the flow.

Understanding these sub-flows is the prerequisite for performance tuning exercises.
Test Simulation

A SoapUI Client was used to simulate unique order generation into the AIA pre-built integration layer. SoapUI Mock services simulated the Siebel CRM and Oracle BRM systems which provide the responses for the Communications O2C Flow.

The following diagram shows the high-level Communications O2C flow as an interaction between the order generator, the test system, and mock services.

Topology: Diagram 3
Baseline Test

The default values for all the tuning knobs were chosen as the starting point for the tuning exercise. The first run was used to record the initial baseline throughput performance of the Communications O2C pre-built integration.

The initial topology chosen for the performance tuning exercise was a 2 node clustered setup. Topology: Diagram 1 illustrates the test topology details.

Each test execution consisted of generating and submitting a fixed number (usually 1000) of sales orders to the system. The TPM was calculated for each test execution based on the successful orders that passed through the test system on average per minute as seen from the composite_instance soainfra table.

The Baseline tests were run with all default parameters. The tuning knobs and their initial default values are listed in the following table:

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>DEFAULT VALUE (OOTB)</th>
<th>COMPONENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resequencer Locker Thread Sleep(sec)</td>
<td>10</td>
<td>Mediator</td>
</tr>
<tr>
<td>Resequencer Maximum Groups Locked</td>
<td>4</td>
<td>Mediator</td>
</tr>
<tr>
<td>Resequencer Worker Threads</td>
<td>4</td>
<td>Mediator</td>
</tr>
<tr>
<td>MDS Connection Pool</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Engine Threads</td>
<td></td>
<td>Weblogic</td>
</tr>
<tr>
<td>ProcessSalesOrderSiebelCommsJMSCConsumer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>adapter.jms.receive.threads</td>
<td>1</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>ProcessSalesOrderSiebelCommsJMSCConsumer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>adapter.jms.receive.timeout</td>
<td>10</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>OrderOrchestrationConsumer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>adapter.jms.receive.threads</td>
<td>1</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>OrderOrchestrationConsumer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>adapter.jms.receive.timeout</td>
<td>10</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>ProcessFulfillmentOrderBillingAccountListOSMCFSComsJMSCConsumer</td>
<td>1</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>adapter.jms.receive.threads</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ProcessFulfillmentOrderBillingAccountListOSMCFSComsJMSCConsumer</td>
<td>10</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>adapter.jms.receive.timeout</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CustomerResponseConsumer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>adapter.jms.receive.threads</td>
<td>1</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>CustomerResponseConsumer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>adapter.jms.receive.timeout</td>
<td>10</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>Component</td>
<td>threads</td>
<td>Timeout</td>
</tr>
<tr>
<td>--------------------------------------------------------------------------</td>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>ProcessFulfillmentOrderBillingOSMCFSCommsJMSCOMConsumer adapter</td>
<td>1</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>adapter.jms.receive.threads</td>
<td>1</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>ProcessFulfillmentOrderBillingOSMCFSCommsJMSCOMConsumer adapter</td>
<td>10</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>adapter.jms.receive.timeout</td>
<td>10</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>Billing ResponseConsumer adapter</td>
<td>1</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>adapter.jms.receive.timeout</td>
<td>10</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>Billing ResponseConsumer adapter</td>
<td>1</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>adapter.jms.receive.timeout</td>
<td>10</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>UpdateSalesOrderOSMCFSCommsJMSCOMConsumer adapter</td>
<td>1</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>adapter.jms.receive.timeout</td>
<td>10</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>JTA</td>
<td>120</td>
<td>Weblogic</td>
</tr>
<tr>
<td>BPEL EJB Timeout</td>
<td>300</td>
<td>BPEL</td>
</tr>
<tr>
<td>Sync Max Wait Timeout</td>
<td>45</td>
<td>BPEL</td>
</tr>
<tr>
<td>Audit Level</td>
<td>Production</td>
<td>SOA INFRA</td>
</tr>
<tr>
<td>Heap</td>
<td>-Xms4096m - Xmx4096m</td>
<td>JVM</td>
</tr>
</tbody>
</table>

With the default settings and no tuning performed, the following baseline Order throughput was observed:

**Total TPM (Transactions per minute) = 9.15**
Adapter Tuning

Two tests were conducted:

1. Number of Orders = 500 and delay between orders (secs) = 5
2. Number of Orders = 1000 and delay between orders (secs) = 2

Observation

We observed build up of messages on AIA_CRTBO_OUT_JMSQ.

Analysis

The property of adapter.jms.receive.threads is set to 1 by default for ProcessFulfillmentOrderBillingOSMCFSCommsJMSConsumer. The JMSConsumer listens on the AIA_CRTBO_OUT_JMSQ and operates in a single threaded fashion. The single thread processes the transaction until the next dehydration point.

We observed that the AIA_CRTBO_OUT_JMSQ queue had message buildup that indicated a bottleneck at this stage of the processing. About 290 messages (out of 500 messages) were piled up on the AIA_CRTBO_OUT_JMSQ at a point during the test execution. To ease this throttling of messages we increased the rate of consumption out of this destination. The TPM was limited by the processing rate of the backlog of these messages so we decided to boost the number of adapter threads for the JMS Consumer from the default value of 1 to a new value of 10.

Note: In a clustered environment, where the JMS Consumer Adapters listen on a uniform distributed destination, the effective number of JMS threads on each node is multiplied by the number of nodes in the cluster. For instance, when the adapter.jms.receive.threads is set to 10, the number of JMS threads listening on each node is actually 20 for a 2 node cluster setup.

The following diagram depicts the buildup of messages in the AIA_CRTBO_OUT_JMSQ as seen from the AIA Queues monitoring data collected during the execution of the test.
Implementation

The adapter.jms.receive.threads property for the Composite ProcessFulfillmentOrderBillingOSMCFSCommsJMSCOnsumer (JCA Component Consume_PFOB) was set to a value of 10 from EM Console.

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>OLD VALUE</th>
<th>NEW VALUE</th>
<th>COMPONENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProcessFulfillmentOrderBillingOSMCFSCommsJMSCOnsumer adapter.jms.receive.threads</td>
<td>1</td>
<td>10</td>
<td>JMS Adapter</td>
</tr>
</tbody>
</table>

Results

Increasing the adapter threads for ProcessFulfillmentOrderBillingOSMCFSCommsJMSCOnsumer effectively removed the bottleneck at the AIA_CRTBO_OUT_JMSQ queue. No pile up of messages was observed at the queue as is evident in the following diagram.
Adapter Tuning: Diagram 2

The tuning also helped to achieve a greater TPM:

**Total TPM = 11.2 (500 Orders)**

**Total TPM = 17.8 (1000 Orders)**

A 22% increase in the TPM value was observed for the Test conducted with 500 orders. Further an improved TPM was observed with 1000 orders test proving that Communications O2C was now able to handle a higher order rate with the removed bottleneck.
Resequencer Tuning

Observation
We observed that the messages spent up to 12 mins in the resequencer tables. The average time for a message to be locked by the resequencer was 53s. Once locked by the resequencer, the messages spent on average about 9 seconds before completion of processing by the resequencer.

Analysis
The SyncCustomerPartyList stage resequenced the messages received from Siebel before calling BRM via SyncCustomerPartyList BRMCommsProvABCSImpl. The latency observed in the resequencer was due to resequencer processing at a lower than optimal rate. The Locker threads were locking groups at a rate slower than the order injection which led to the pileup of messages at the resequencer stage. For a faster processing rate, the resequencer required tuning.

We tuned the following resequencer parameters to improve the performance of the resequencer and eliminate the bottleneck at this stage:

1. Resequencer Locker Thread Sleep(sec) – default value 10s
   This parameter controls the amount of time the locker thread sleeps (in seconds) between iterations when no groups are found for locking.

2. Resequencer Maximum Groups Locked – default value 4 groups
   Defines the maximum number of groups to be locked in each iteration by the locker thread.

3. Resequencer Worker Threads – default value 4 threads
   Defines the number of worker threads that process the locked groups. Note that only one worker thread processes messages for a locked group.

These parameters directly impact the amount of time spent by the messages waiting to be locked.

Implementation
The resequencer parameters were increased gradually from the default values while we observed their correlation with the average time spent in the resequencer.

The Resequencer Worker Threads were increased from 4 to 16, then to 32, and finally to 50 threads.

The Resequencer Locker Thread Sleep was reduced from 10s to 1.

Because a large number of groups were expected during the test runs, we changed the value for Resequencer Maximum Groups Locked from 4 to 100.

The old and new values of the parameters are shown in the following table:
<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>OLD VALUE</th>
<th>NEW VALUE</th>
<th>COMPONENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resequencer Locker Thread Sleep(sec)</td>
<td>10</td>
<td>1</td>
<td>Mediator</td>
</tr>
<tr>
<td>Resequencer Maximum Groups Locked</td>
<td>4</td>
<td>100</td>
<td>Mediator</td>
</tr>
<tr>
<td>Resequencer Worker Threads</td>
<td>4</td>
<td>50</td>
<td>Mediator</td>
</tr>
</tbody>
</table>

**Results**

The tuning resulted in great improvements in the resequencer performance.

The average time waiting for resequencer locking was 23s (down from 53s) and the average time spent by the resequencer processing the locked messages was 5s (down from the 9s).

Overall this resequencer tuning had an impact of reducing the bottleneck at this stage and boosted the order throughput rate by about 84%.

**Total TPM = 32.89 (1000 Orders)**
Adapter Tuning (Increase Threads on all Consumers)

Observation

We observed that messages piled up in various upstream and downstream processing queues.

Analysis

After the tuning done to ProcessFulfillmentOrderBillingOSMCFSCommsJMSComsumer, we observed that the other JMSConsumers were not able to keep up with the same message processing rate as the tuned JMSConsumers. Through various tests of 1000 orders, messages were piling up at the following queues:

AIA_SALESORDERJMSQUEUE (ProcessSalesOrderFulfillmentSiebelCommsJMSComsumer)
AIA_CRTCUST_OUT (ProcessFulfillmentOrderBillingAccountListOSMCFSCommsJMSComsumer)
AIA_UPDCUST_IN_JMSQ (CustomerResponseConsumer)
AIA_UPDSO_OUT (UpdateSalesOrderOSMCFSCommsJMSComsumer)

The following graphs show the pattern of messages pileup in the AIA queues. Note that the messages initially piled up at the AIA_SALESORDERJMSQUEUE and as the messages were picked up for processing, they piled up to a smaller extent in downstream queues AIA_CRTCUST_OUT, AIA_UPDCUST_IN_JMSQ. Finally we observed that further downstream queue AIA_UPDSO_OUT experienced small bursts of messages as the messages were released from upstream processing.
This pattern clearly showed that the JMSConsumers listening on these queues were not able to keep up with the rate of processing of messages. As discussed in “Adapter Tuning” the remaining
JMSConsumers must be tuned so that the rate of processing matches the ProcessFulfillmentOrderBillingOSMCFSCommsJMSConsumer.

Subsequently the value of adapter.jms.receive.threads for the remaining JMSConsumers was changed from the default value of 1 to a new value of 10. We observed that the consumers that receive multiple messages per order need to be pegged at a higher thread value to match the overall processing rate.

**Implementation**

The value for adapter.jms.receive.threads for the following JMSConsumers were increased to values as shown in this table:

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>OLD VALUE</th>
<th>NEW VALUE</th>
<th>COMPONENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProcessSalesOrderSiebelCommsJMSConsumer adapter.jms.receive.threads</td>
<td>1</td>
<td>10</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>OrderOrchestrationConsumer adapter.jms.receive.threads</td>
<td>1</td>
<td>10</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>ProcessFulfillmentOrderBillingAccountListOSMCFSCommsJMSConsumer adapter.jms.receive.threads</td>
<td>1</td>
<td>80</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>CustomerResponseConsumer adapter.jms.receive.threads</td>
<td>1</td>
<td>10</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>ProcessFulfillmentOrderBillingOSMCFSCommsJMSConsumer adapter.jms.receive.threads</td>
<td>10</td>
<td>80</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>Billing ResponseConsumer adapter.jms.receive.threads</td>
<td>1</td>
<td>10</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>UpdateSalesOrderOSMCFSCommsJMSConsumer adapter.jms.receive.threads</td>
<td>1</td>
<td>40</td>
<td>JMS Adapter</td>
</tr>
</tbody>
</table>

**Results**

No pile up of messages was observed in the Communications O2C pre-built integration flow queues following the changes to the adapter.jms.receive.threads settings to the corresponding JMSConsumers. The following diagram shows almost no pileup in the tuned queues.
These queues now enabled a higher throughput of the system. The new observed value of Order Throughput per minute was 48% higher.

Total TPM = 48.91 (1000 Orders)
Increase Order Rate

Observation

We observed that the order throughput was limited by the input rate of the order injection simulation.

Analysis

In the previous section, the bottlenecks at various queues were eliminated by tuning the corresponding JMSConsumers. To further improve the throughput of the system there was a need to increase the rate of order input into the system.

Up until this point, the medium sized orders were fed into the AIA_SALESORDERJMSQUEUE with a delay of 1s between orders, while the JMSConsumer remaining active during the order injection process.

In order to increase the order injection rate, we preloaded the AIA_SALESORDERJMSQUEUE queue with orders with the listening Consumer switched off and then flooded the system with the preloaded orders.

Implementation

The SoapUI order injection client’s order delay was reduced to 0s. The ProcessSalesOrderFulfillmentSiebelCommsJMSConsumer state was turned off from EM console and 1000 orders were delivered to the destination AIA_SALESORDERJMSQUEUE. Once all the orders were in place, the JMSConsumer was switched ON.

In addition the SoapUI client was moved from one of the application nodes to a separate host. This ensured that there was no load asymmetry between the application server nodes owing to the test client.

Results

The increased order input rate increased the throughput by 45.5%.

**Total TPM = 71.2 (1000 Orders)**
Memory Tuning

Observation

We observed frequent and back-to-back full GC cycles on both the app node JVMs.

Analysis

We observed frequent full GCs from the initial OOTB test. The starting value of 4GB total heap size was inadequate for Communications O2C. This was indicated by the garbage collections triggered by periodical filling up of the heap and the free up of memory following the GCs. The heap allocation was incrementally raised to 8GB and then 12GB which was found to be sufficient.

This section discusses the further JVM tunings that were performed with the allocated heap size of 12GB.

With 12GB total heap allocated, Communications O2C pre-build integration queues tuned, and order rate increased we observed that the JVMs were executing back-to-back full garbage collections.

The following diagrams are GCViewer charts from GC logs collected from appNode1 and appNode2 during the test execution. The used heap and Full GC pause times are shown in the chart. During the test execution, we observed that, back-to-back full GCs were running on both the nodes resulting in degraded performance and many timeout exceptions. Notable among the timeouts were JTA transaction timeouts and Resequencer store errors which coincided with the back to back full GCs observed on both appNodes.
Memory Tuning: Diagram 1

Memory Tuning: Diagram 2
For the Garbage collector scheme we used the throughput collector, configured using the ‘UseParallelOldGC’. This used parallel GC threads during both the young collection and tenured collection phases.

The aim was to tune the JVM to reduce the full GC pause times and also reduce the overall number of full GCs during the test execution.

The New Generation size was tuned to be approximately 60% of the total heap allocation. The number of ParallelGCThreads used for Old Generation collection was increased from 8 to 16. We expected a reduction of the GC pause times due to the increased parallel collections. Also the Survivor ratio was reduced from 6 to 2 and TargetSurvivorRatio was set to 90% to give bigger survivor spaces during the copy phases and delay the promotion of objects to the tenured space.

The Hotspot code cache parameters were tuned in response to the following warning displayed during the JIT compilations.

‘Java HotSpot(TM) 64-Bit Server VM warning: CodeCache is full. Compiler has been disabled’

The JVM attributes -XX:ReservedCodeCacheSize=128m -XX:+UseCodeCacheFlushing were added to provide sufficient code cache size.

Finally we enabled the Oracle Solaris libumem library to take advantage of the multithread aware memory allocation on Sparc T4-2 environment.

### Implementation

The following table lists the JVM parameters added as part of the JVM tuning:

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>OLD VALUE</th>
<th>NEW VALUE</th>
<th>COMPONENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemAlloc Library</td>
<td>N/A</td>
<td>LD_PRELOAD=libumem.so</td>
<td>OS</td>
</tr>
</tbody>
</table>
Results

The JVM Tuning resulted in reduced average GC pause times and reduced number of full GCs. The following charts show the worst case full GC pause times with the new JVM parameters. Notable changes included an almost 100% reduction in full GC cycle pause times and the ability of the system to recover from the full GCs and continue with the processing.

Memory Tuning: Diagram 3
Memory Tuning: Diagram 4

The GC Tuning resulted in an increased order throughput of 34%:

Total TPM = 95.6 (1000 Orders)
Vertical Scaling of SOA Nodes

Observation

The average CPU Utilization as monitored during the test run was lower than optimal. Although the peaks of CPU Utilization reached about 80% at times during the test runs, on average the load on the CPU was about 22% during a typical test execution. We decided to scale out the system to 4 SOA nodes to further utilize the available processing power.

![CPU Utilization before Scaleout](image)

Analysis

The system was scaled out by adding 2 additional managed servers for SOA and configuring AIA and the pre-built integration for the new nodes.

Topology: Diagram 2 illustrates the resulting system topology.

Implementation

We reduced the heap allocation per SOA server JVM to 10GB (from the earlier 12GB) to accommodate the additional 2 JVMs after scale-out within the available memory in the Physical host. However, the ratios of New Size, Survivor spaces, tenured space, and PermGen space were maintained at the earlier 2-node configurations.

The local disk used for JMS Filestore presented a performance bottleneck after the scale-out to 4 servers. Iostat monitoring showed that the disk was 96% busy and consuming about 150+
milliseconds per request. To alleviate the IO bottleneck, a faster SSD disk was locally mounted and used for JMS Filestore. As a result the IO write times per request were restored to less than 10 milliseconds. We also moved the temp directory IO to the faster SSD disk by specifying the JVM parameter “-Djava.io.tempdir”.

The following table shows the JVM parameters changed as part of the scale-out:

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>OLD VALUE</th>
<th>NEW VALUE</th>
<th>COMPONENT</th>
</tr>
</thead>
</table>

Results

As a result of the SOA node scale-out to 4 servers, the average CPU utilization during test execution more than doubled to approximately 59%.
Scale-out: Diagram 2

CPU Utilization after Scale-out

The new order throughput:

Total TPM = 163 (2000 Orders)
Adapter Tuning (Decrease threads)

Observation

When Adapter threads were configured with the following values, the tests failed.

<table>
<thead>
<tr>
<th>ADAPTER.JMS.RECEIVE.THREADS</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProcessSalesOrderSiebelCommsJMSConsumer</td>
<td>10</td>
</tr>
<tr>
<td>OrderOrchestrationConsumer</td>
<td>10</td>
</tr>
<tr>
<td>ProcessFulfillmentOrderBillingAccountListOSMCFSCommsJMSC</td>
<td>80</td>
</tr>
<tr>
<td>CustomerResponseConsumer</td>
<td>10</td>
</tr>
<tr>
<td>ProcessFulfillmentOrderBillingOSMCFSCommsJMSCConsumer</td>
<td>80</td>
</tr>
<tr>
<td>Billing ResponseConsumer</td>
<td>10</td>
</tr>
<tr>
<td>UpdateSalesOrderOSMCFSCommsJMSCConsumer</td>
<td>40</td>
</tr>
</tbody>
</table>

The tests failed with few orders not making it through the end-to-end flow.

Analysis

We observed a lot of back-to-back full GCs in the GC logs, during the test run with the above thread values. We also observed that once a node went into back to back full GC, it couldn’t recover from the cycle of full GCs. Occasional full GC is not sign of concern. However, if the back-to-back full GCs occur during transaction processing and they continue for the duration of the transaction; then this may be a sign of a problem. We noticed that some order were lost during the test. We could attribute the loss of orders to the back-to-back full GCs because all orders completed the end to end flow when order injection rate was reduced. Because all the orders completed the end-to-end flow with a lower order injection rate, we suspected that throttling to be the cause of back-to-back full GCs. The throttling can be achieved either altering the order injection rate or tuning the consumer threads in the Communication O2C pre-built integration flow. Since the goal was to get maximum order throughput, the order injection rate should not be modified. Hence we concluded that due to the high thread count of the consumer, there were too many transactions that caused resource contention and high memory utilization. This led to trigger back-to-back full GCs.

The analysis led us to further tune the consumer threads but this time to decrease the threads to avoid resource contention and high memory utilization.

Implementation

To arrive at the optimal number of consumer threads, we ran several tests and varied the thread numbers in every test. The threads were tuned with both 2 nodes up and 4 nodes up.
This table shows the values used for each parameter that produced the best performance:

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>OLD VALUE</th>
<th>NEW VALUE</th>
<th>COMPONENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProcessSalesOrderSiebelCommsJMSCConsumer adapter.jms.receive.threads</td>
<td>10</td>
<td>4</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>OrderOrchestrationConsumer adapter.jms.receive.threads</td>
<td>10</td>
<td>4</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>ProcessFulfillmentOrderBillingAccountListOMCFSCommsJMSCConsumer adapter.jms.receive.threads</td>
<td>80</td>
<td>4</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>CustomerResponseConsumer adapter.jms.receive.threads</td>
<td>10</td>
<td>4</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>ProcessFulfillmentOrderBillingOSMCFSCommsJMSCConsumer adapter.jms.receive.threads</td>
<td>80</td>
<td>4</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>Billing ResponseConsumer adapter.jms.receive.threads</td>
<td>10</td>
<td>4</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>UpdateSalesOrderOSMCFSCommsJMSCConsumer adapter.jms.receive.threads</td>
<td>40</td>
<td>4</td>
<td>JMS Adapter</td>
</tr>
</tbody>
</table>

Results

The following chart indicates the TPM achieved by varying the count of consumer threads on 2 managed server nodes running on different machines.
The following chart indicates the TPM achieved by varying the count of consumer threads on 4 managed server nodes running on different machines.
Total TPM = 174

**Best Practice:** Do not increase any tuning knob (in this case the threads) drastically. In this exercise the consumer threads were bumped up in the preceding tests. The high thread numbers provided good performance improvement for the initial tests but in the later tests with increased order rate, the high thread count deteriorated the performance.
Database Tuning

Observation

Since the database can easily become the bottleneck in SOA/AIA deployments, we recommend that you run database performance statistic reports to analyze the behavior of the SOA infrastructure database. Components such as BPEL and Mediator may make heavy use of the database depending on the nature of the services and their configuration.

Oracle database server provides various reports such as the AWR and the ADDM reports. These reports are implemented as SQL scripts (awrrpt.sql and addmrpt.sql) and can be found in directory <ORACLE_HOME>/rdbms/admin.

While both reports rely on the same set of database statistics held in the workload repository, the ADDM report has more of an advisor view highlighting most relevant areas while the AWR report provides a full list of all performance details.

The database EM console indicated some hardware contention on LOB tables during load.

![DB Tuning: Diagram 1](image)

Analysis

The brown peaks in the DB Tuning Diagram 1 indicated waits for class “Configuration”. More detailed analysis (by clicking on the wait class in the diagram or by analyzing the AWR report) showed “High Watermark Contention” on some of the LOB tables (see below). Oracle secure files
feature, introduced many enhancements for improved performance, efficient storage, and easier manageability.

**Top 5 Timed Foreground Events**

<table>
<thead>
<tr>
<th>Event</th>
<th>Waits</th>
<th>Time(s)</th>
<th>Avg wait (ms)</th>
<th>% DB time</th>
<th>Wait Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>eq: HW - contention</td>
<td>20,935</td>
<td>3.195</td>
<td>153</td>
<td>27.84</td>
<td>Configuration</td>
</tr>
<tr>
<td>DB CPU</td>
<td>2,805</td>
<td></td>
<td></td>
<td>24.26</td>
<td></td>
</tr>
<tr>
<td>db file sequential read</td>
<td>250,767</td>
<td>1.485</td>
<td>6</td>
<td>12.85</td>
<td>User I/O</td>
</tr>
<tr>
<td>gc or block 2-way</td>
<td>2,296,979</td>
<td>884</td>
<td>0</td>
<td>7.47</td>
<td>Cluster</td>
</tr>
<tr>
<td>gc buffer busy release</td>
<td>42,105</td>
<td>632</td>
<td>15</td>
<td>5.47</td>
<td>Cluster</td>
</tr>
</tbody>
</table>

**Implementation**

To implement secure files for SOA-infra tables refer to the following documentations:

   a. If SecureFiles needs to be enabled for SOA-infra tables that are created newly then refer to section “Using SecureFiles for new systems” under “How to enable SecureFiles for the Oracle FMW SOA Schemas”.
   b. If secure files needs to be enabled for already existing SOA-infra tables then refer to section “Using Secure Files for existing/running systems” under “How to enable SecureFiles for the Oracle FMW SOA Schemas”

2. [http://docs.oracle.com/cd/B28359_01/appdev.111/b28393/adlob_smart.htm#BABDIEGE](http://docs.oracle.com/cd/B28359_01/appdev.111/b28393/adlob_smart.htm#BABDIEGE)


**Results**

After implementing the Secure Files, the brown waits of class “Configuration” were removed as shown in the following chart.
The secure file implementation, along with purging records in the database, provided a marginal improvement in the performance.

**Throughput Gain = 3%**

**Total TPM = 179**
Audit Tuning

Observation

By default the audit trail of a SOA composite is stored synchronously in the same thread of the transaction, which adds latency for the end-to-end transactions. You can tune the audit store using the auditStorePolicy parameter. For generic guidelines for tuning all audit parameters, refer to “Tuning Timeouts and Audit Level”.

Analysis

Turning off the Audit (Audit Level = off) gives the best performance because no data gets written to the database; however, turning off the Audit logging may not be an ideal scenario. This may be an option for very stable production environments where the requirement is a very high throughput. Most use cases involve tuning of the audit parameters.

When you set Audit Level=Development all data of the transaction is written into the database. With auditStorePolicy=syncSingleWrite, the audit gets written synchronously in the same thread of the transaction. This adds latency for the end-to-end transaction.

Although the performance exercise did not cover tuning of different Audit Parameters to get the throughput, it can be logically concluded that having Audit Level=Development with auditStorePolicy=syncSingleWrite gives the least performance benefit. Tuning Audit Level and auditStorePolicy appropriately can have better performance benefits, but for the maximum benefit, set Audit Level=off.

It should be noted that the audit level settings in SOA 11g are at multiple levels (Composite Level, BPEL Level, and Mediator Level). By default, if the composite audit level is changed from development to production, the audit level of its components is changed as well. This is because the Audit Level is inherited from the composite. But if the Audit Level of a BPEL process is changed from development to production, the composite level remains unchanged.

In SOA 11g there is no Audit Level feature to write the composite instance only when the composite fails. This is only available with BPEL with completionPersistPolicy that is discussed in a previous section. This means that the composite Audit Level can only be turned off completely or kept on (“Development” or “Production”). For more detailed analysis on the performance of different audit levels, refer to the white paper on http://stefankoser.blogspot.de/2012/11/new-whitepaper-soa-11g-influence-of.html.
Audit Tuning: Diagram 1

Implementation

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>OLD VALUE</th>
<th>NEW VALUE</th>
<th>COMPONENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audit Level</td>
<td>Production</td>
<td>Off</td>
<td>SOA Infra</td>
</tr>
</tbody>
</table>

Tuning other Audit Parameters is out of scope of this document.

Result

By turning the Audit Level = off the following was achieved -

Throughput Gain = 29%

Total TPM = 230
Response Time Analysis

During the performance tuning of the Communications O2C pre-built integration flow, the emphasis was on maximizing the order throughput. But it is important to observe the ‘Average Response Time’ of the system during the course of the tuning. For this discussion, Average Response Time is the time consumed an order takes to complete the Communications O2C pre-built end-to-end flow. The following diagram shows the graph of Order Throughput (TPM) and Average Response time plotted for the milestone tests discussed in the preceding sections.

The Average Response Time came down drastically as part of the tuning, from 397 seconds in the initial test to about 125 seconds with the final tuned configuration. However for some of the tuning milestones, the response time went up to about 499 seconds.

The response time increase was seen in two milestones:

- When the JMS adapter threads of the consumers were increased. This was due to increased contention between the increased numbers of threads for order processing.
- When the order injection rate was increased. The increase in order injection rate further augmented the resource contention and caused a degraded response time.

Finally, we saw that with appropriate memory tuning and further fine tuning the consumer threads, the system was able to handle the higher order injection rate. The result was a reduced response time and increased order throughput.

When tuning an AIA system for throughput, there may be times when the response time takes a hit, so it is recommended to monitor the Average Response Time during performance tuning. It is essential to reach a balance by further fine tuning, in order to get the best performance out of the system.
Summary

The final configuration for the Asynchronous flow is shown in the following table:

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>DEFAULT VALUE (OOB)</th>
<th>COMPONENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resequencer Locker Thread Sleep (sec)</td>
<td>1</td>
<td>Mediator</td>
</tr>
<tr>
<td>Resequencer Maximum Groups Locked</td>
<td>100</td>
<td>Mediator</td>
</tr>
<tr>
<td>Resequencer Worker Threads</td>
<td>50</td>
<td>Mediator</td>
</tr>
<tr>
<td>MDS Connection Pool</td>
<td>100</td>
<td>Weblogic</td>
</tr>
<tr>
<td>Engine Threads</td>
<td>100</td>
<td>BPEL</td>
</tr>
<tr>
<td>ProcessSalesOrderSiebelCommsJMSCConsumer adapter.jms.receive.threads</td>
<td>4</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>ProcessSalesOrderSiebelCommsJMSCConsumer adapter.jms.receive.timeout</td>
<td>1000</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>OrderOrchestrationConsumer adapter.jms.receive.threads</td>
<td>4</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>OrderOrchestrationConsumer adapter.jms.receive.timeout</td>
<td>1000</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>ProcessFulfillmentOrderBillingAccountListOSMCFSCommsJMSCConsumer adapter.jms.receive.threads</td>
<td>4</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>ProcessFulfillmentOrderBillingAccountListOSMCFSCommsJMSCConsumer adapter.jms.receive.timeout</td>
<td>1000</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>CustomerResponseConsumer adapter.jms.receive.threads</td>
<td>4</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>CustomerResponseConsumer adapter.jms.receive.timeout</td>
<td>1000</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>ProcessFulfillmentOrderBillingOSMCFSCommsJMSCConsumer adapter.jms.receive.threads</td>
<td>4</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>ProcessFulfillmentOrderBillingOSMCFSCommsJMSCConsumer adapter.jms.receive.timeout</td>
<td>1000</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>Billing ResponseConsumer adapter.jms.receive.threads</td>
<td>4</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>Billing ResponseConsumer adapter.jms.receive.timeout</td>
<td>1000</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>UpdateSalesOrderOSMCFSCommsJMSCConsumer adapter.jms.receive.threads</td>
<td>4</td>
<td>JMS Adapter</td>
</tr>
<tr>
<td>Parameter</td>
<td>Value 1</td>
<td>Value 2</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>----------------</td>
<td>----------------</td>
</tr>
<tr>
<td>JTA</td>
<td>360</td>
<td>Weblogic</td>
</tr>
<tr>
<td>BPEL EJB Timeout</td>
<td>300</td>
<td>BPEL</td>
</tr>
<tr>
<td>Sync Max Wait Timeout</td>
<td>250</td>
<td>BPEL</td>
</tr>
<tr>
<td>Audit Level</td>
<td>off</td>
<td>SOA INFRA</td>
</tr>
</tbody>
</table>

The tuning done over multiple tests provided performance increments as shown in the following graph:
Tuning Results Overview

- Baseline: 9
- First Adapter Tuning: 11
- Resequencer Tuning: 33
- Scheduling Tuning: 49
- JVM GC Tuning: 96
- Scale out to 4 nodes and I/O tuning: 163
- Optimizing JMS consumer threads: 174
- DB Tuning, Secure Files: 179
- AuditLevel=Off: 230

Orders per Minute
Performance Tuning of Synchronous Flows

Since the focus of the performance tuning exercise was the asynchronous integration flows, the synchronous flow was not tuned for maximum performance. It has been included here for sake of completeness. The Oracle AIA Agent Assisted Billing Care (AABC) synchronous flow was chosen for this exercise.

Oracle AIA AABC Pre-built Integration Flow

The AABC pre-built integration consists of online requests originating from Siebel CRM. These requests fetch the responses synchronously from Oracle BRM system using the synchronous integration patterns offered by the AABC flow. AABC employs AIA Connector services and Enterprise Business Services to implement the integrations.

The following diagram depicts the high-level AABC flows.

![Diagram of AABC synchronous integration flows]

Test Simulation

A jMeter Client was used to simulate load generation from Siebel into the AIA pre-built integration layer. SoapUI Mock services simulated the Siebel and BRM systems which provide the responses for the AABC flow.
The following figure shows the high level AABC flow as an interaction between the load generator, the test system, and mock services.

![AABC Flow Diagram]

Test Results

The following table shows the throughput and average response time for AABC tests for the OOTB settings and with the audit level switched from “Development” to “Production”.

<table>
<thead>
<tr>
<th>TEST RUN</th>
<th>AABC THROUGHPUT (PER SECOND)</th>
<th>AVG TIME (SECONDS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. OOTB Test Run</td>
<td>14.2</td>
<td>3.4</td>
</tr>
<tr>
<td>2. Test run with Audit Level=Production</td>
<td>16.6</td>
<td>3.0</td>
</tr>
</tbody>
</table>

The BRM simulator contributed a fixed latency of 2.8 seconds to the response time. The actual improvement in average response time of the AIA integration was a significant 0.4 seconds down from 0.6 seconds.

More tuning of the synchronous process can be achieved following the synchronous process tuning knobs specifically and tuning knobs guidelines in general.
Conclusion

We have shown that by systematically tuning the various layers of an AIA system, it can be configured for high performance goals. In the asynchronous performance tuning exercise, the order throughput was increased from 9.13 TPM in the baseline test to a final value of 230 TPM. It was also observed that the system could have degraded performance when the knobs are not tuned judiciously.

The underlying server platform also forms the vital infrastructure and provides the backbone for a high performance system. The Oracle SPARC T4 servers provide a solid platform to run AIA and SOA-based integrations. The combination of high single-threaded performance and throughput make them ideal for middleware workloads. As a general statement for good all-round performance, the SPARC T4 servers are recommended for all AIA/SOA workloads.

The general guidelines provided in this document to tune AIA pre-built integrations may be applied to any AIA or even non-AIA based integration systems that employ Oracle FMW components.
Appendix A: SQL Queries used for Monitoring

The following are sample sql queries used for monitoring the performance test executions. They are examples and you will need to make suitable changes for any other system.

**Communications O2C Order Throughput Measurement: Transactions per minute Query:**
```
select min(created_time) start_time, max(created_time) completion_time, max(created_time) -
min(created_time) time ,
count(1)/9 orders, count(*)/((to_number(extract(hour from (max(created_time) -
min(created_time)))))*60+to_number(extract(minute from (max(created_time) -
min(created_time))))*60+to_number(extract(second from (max(created_time) -
min(created_time)))))/60))/9 tp_per_min
from AIAPERF_SOAINFRA.composite_instance
where title between ?1 and ?2;
```

The range of order numbers used for a particular test execution was plugged into the previous query. The title for the test orders was 'SalesOrder - MEDIUM-15656891-326999'

**Communications O2C Order Average Response Time Query:**
```
select trunc(avg((sysdate+ (MAX(co.created_time) -
MIN(co.created_time))*24*60*60 -
systime)),2) avg_Processing_Time
from composite_instance co
where co.title like '%MEDIUM%'
and title between ?1 and ?2
union
select to_char(sysdate,'YYYY-MM-DD HH24:MI:SS') time, 'Sync CustomerParty ' sequencer,
gs.status grou
up_status, m.status msg_status, count(1)
from AIAPERF_SOAINFRA.mediator_group_status gs,
AIAPERF_SOAINFRA.mediator_resequencer_message m
where m.group_id = gs.group_id
and gs.component_dn =
'default/CommunicationsCustomerPartyEBSV2Resequencer!1.0/CommunicationsCustomerPartyEB
SV2Resequencer'
and gs.status < 3
union
select to_char(sysdate,'YYYY-MM-DD HH24:MI:SS') time, 'Update SalesOrder' sequencer, gs.status
踪group_status, m.status msg_status, count(1)
from AIAPERF_SOAINFRA.mediator_group_status gs,
AIAPERF_SOAINFRA.mediator_resequencer_message m
```
where m.group_id = gs.group_id
and gs.component_dn =
'default/UpdateSalesOrderOSMCFSCommsJMSConsumer!1.0/Consume_UPDSO_RS'
and gs.status < 3
group by gs.status, m.status

**Communications O2C Resequencer lock_phase and work_phase times Query**

```sql
select g.lock_time,
       trunc(((sysdate+(g.lock_time-m.creation_date)*24*60-sysdate)+420)*60,2))
       lock_time_in_secs,
       trunc(((sysdate+(g.last_received_time-g.lock_time)*24*60-sysdate)-420)*60,2)
       work_time_in_secs
from mediator_resequencer_message m, mediator_group_status g
where g.lock_time between TO_DATE('10/11/2012 19:25:00', 'MM/DD/YYYY hh24:mi:ss') and
     TO_DATE('10/10/2013 15:12:28', 'MM/DD/YYYY hh24:mi:ss')
and g.group_id=m.group_id
and m.component_dn like '%CommunicationsCustomerPartyEBSV2Ressequencer%'
order by g.lock_time
```

Start and end times of the test Execution are plugged into the previous query
Appendix B: Tuning Oracle JRockit

Oracle JRockit is optimized for Intel architecture (x86, x86-64), provides better runtime performance and management capability than Hotspot on such hardware. On Oracle SPARC servers running Oracle Solaris, Hotspot is the preferred JVM, while on other platforms using Intel architecture, JRockit is recommended.

Tuning Memory Footprint
Since Oracle JRockit uses generational heap as shown in the following diagram, the basic setting for JRockit is to determine the right size for the Nursery and Tenured spaces.

![Diagram of JVM memory footprint](image)

The general guidelines for tuning the JVM memory are as follows:

- Use larger heaps whenever possible on modern servers (64 bit): For example, use 12g or 16g as the default configuration for `-Xmx`, and then resize it as needed.
- Set the initial heap size equal to the maximum size. E.g., `-Xms:12g -Xmx:12g`
- Set the Nursery space specified by `-Xns` to at least 40% of total heap. To prevent frequent or immature object promotion from the Nursery to the Tenured space, setting a larger Nursery (50% or 60% of total heap) is the best practice for the SOA/AIA environment. For example, `-Xms:12g -Xmx:12g -Xns:6g`

Enabling GC logs

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>DEFAULT VALUE</th>
<th>GUIDELINES</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-Xverbose:gc</code></td>
<td></td>
<td>It is highly recommended that you enable GC logs on all environments (Development, Testing, and Production). The performance overhead of logging is minor and can be ignored.</td>
</tr>
<tr>
<td><code>-XverboseTimeStamp</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>-XverboseLog:&lt;location&gt;</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tuning GC
The main goal of tuning GC is to specify the right garbage collection strategy. JRockit decides the garbage collection strategy based on priorities as follows:
- Throughput (default): This priority is specified by using `-Xgc:throughput`, which maximizes throughput at the cost of lower latency. This priority is recommended for an SOA/AIA environment.

- Pause-time: This priority is specified by `-Xgc:pausetime` which optimizes for the length of each garbage collection pause, where latency/response time is more important, e.g., an online transaction processing system.

### Tuning Other Parameters

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>DEFAULT VALUE</th>
<th>GUIDELINES</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-XX:+HeapDumpOnOutOfMemoryError</code></td>
<td></td>
<td>With these parameters, the heap dump will be generated when Out Of Memory error happens.</td>
</tr>
<tr>
<td><code>-XX:HeapDumpPath=&lt;location&gt;</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
