About this White Paper ................................................................................. 3
Who should read this document .................................................................... 3
Before You Begin ..................................................................................................... 3
ADF Overview ..................................................................................................... 1
Application Overview .......................................................................................... 1
Oracle Solaris 10 ............................................................................................... 2
  Sun SPARC Enterprise T-Series Servers ............................................................... 2
  Sun SPARC Enterprise M-Series Servers ............................................................... 2
Workload Overview ............................................................................................ 2
Configuration ........................................................................................................... 3
  Software Configuration ....................................................................................... 3
  Hardware Configuration ....................................................................................... 4
Tuning Recommendations ..................................................................................... 5
  Oracle Solaris Network ....................................................................................... 5
  Oracle RDBMS .................................................................................................... 5
  WebLogic Server ............................................................................................... 6
  Oracle Java ......................................................................................................... 6
  Garbage Collection ............................................................................................ 6
  Baseline testing ................................................................................................. 7
  JVM Tuning ........................................................................................................ 9
Conclusion ........................................................................................................... 11
References......................................................................................................... 12
About this White Paper

This paper describes best practices during deployment of a typical Java Platform, Enterprise Edition (Java EE) application that uses Oracle Application Development Framework (Oracle ADF) Business Components and Oracle ADF Faces. The paper specifically looks at: optimizing resource utilization; improving performance in terms of throughput and response times, and scaling horizontally on Oracle Chip Multi-Threading systems such as the T5240.

Who should read this document

Developers and System Administrators who are either developing, deploying or managing Oracle Fusion Middleware Applications that make use of Oracle ADF components will greatly benefit from this guide.

We assume that the reader is familiar with developing Java EE applications, Oracle ADF development and WebLogic Administration features.

Before You Begin

Although the tuning options provided are specific to the Fusion Order Demo (FOD) application that was tested, detailed notes on why a particular option needs to be monitored and optimized for best performance is provided. The tuning options are specific to the FOD application that has similar load characteristics to an online store. Hence, applying these tunings to your applications without properly measuring the impact for your load may not help.
ADF Overview

Oracle Application Development Framework (Oracle ADF) is an innovative, yet mature Java EE development framework which is directly supported and enabled by the latest in a series of development environments, Oracle JDeveloper 11g. Oracle ADF simplifies Java EE development by minimizing the need to write code that implements the application’s infrastructure allowing the users to focus on the features of the actual application. Oracle ADF provides these infrastructure implementations as part of the framework. Extending beyond recognizing a set of runtime services, Oracle ADF is also focused on the development experience to provide a visual and declarative approach to Java EE development through the Oracle JDeveloper 11g development tool.

Application Overview

The Fusion Order Demo (FOD) is a Store Front demonstration application that allows users to browse, search and purchase various products. It makes extensive use of most Oracle ADF features. Fusion Order Demo comes in two main parts, the StoreFrontModule which is a Web application built on Oracle ADF components and includes all of the front end components, and a bundle of Composite Services that are used to process orders generated by the StoreFrontModule. The testing performed for this paper used only the StoreFrontModule.

The StoreFrontModule consists of two components:

• StoreFrontService: This component provides transactional and secure access to the data stored in the StoreFront Database

• StoreFrontUI: This component provides all of the Web Pages the user uses to browse and search products, and to place orders.

StoreFrontUI uses JavaServer Faces to generate views and leverages the Oracle ADF Data Model layer to interact with Oracle ADF Business Components within the StoreFrontService.

The StoreFrontModule can be deployed as an Enterprise Application Archive (EAR) file or deployed directly from JDeveloper. The latter was chosen for this test as it is recommended that for optimal performance the application MetaData be deployed to an RDBMS based MetaData repository and not as a MetaData Archive deployed with the application EAR file.

The Fusion Order Demo JDeveloper Projects include an Infrastructure project that can be used to setup the RDBMS and to populate it with sample data as well as sample images. Only the sample data was used for testing described in this paper.

Given that the backend Composite Services weren't being used and that the number of available users was limited to those provided by the sample data, testing was limited to operations that did not require a user to login. These operations are described in the Workload Overview section of this document.
Oracle Solaris 10

Oracle Solaris 10 is the Enterprise Operating Environment for the Sun SPARC Enterprise server line. In addition to being the reference platform for many Java SE and Java EE implementations, it provides a number of unique features not found elsewhere:

- ZFS Filesystems, providing 128-bit volumes, double-bit parity, and enhanced ease of administration
- DTrace Observability Framework, providing non-intrusive debugging and observability down into Solaris itself

Sun SPARC Enterprise T-Series Servers

The T-Series servers utilize the multi-threaded UltraSPARC T-Series CPUs to provide the ideal throughput environment for Web Tier Applications.

Sun SPARC Enterprise M-Series Servers

M-Series servers, powered by SPARC64 VII CPUs, are well-suited for running Oracle Databases for application back-ends due to their excellent single threaded performance.

Workload Overview

When developing workloads for testing web applications, user interactions with the application are represented by Operations. Each operation has the following characteristics:

- Set of HTTP requests and responses
- Encompasses events occurring between the client and the server during a single user interaction with the application.

For the Oracle ADF workload several operations were considered. Due to the nature of ADF and of the application some Operations were required to encompass more than a single user interaction with the application. An example of this is the search operation which in FOD requires 5 HTTP requests which have to be run consecutively. Because of this it was decided to split the search operation into two separate operations running consecutively: SearchPanel and SearchRun. The descriptions for the Search operations are found in Table 1.

The other Operations are HomePage and HomePageAddItem. HomePage is the entry point into the application and effectively resets the state of the application for that user on the server. HomePageAddItem is used to add an item to the shopping cart from the home page.
Tuning Oracle ADF applications on Sun T-series Servers

### Table 1: Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HomePage</td>
<td>Makes an initial GET request to the server for the StoreFront and builds the POST request needed to retrieve the HomePage. Manually follows redirects to complete the Operation.</td>
</tr>
<tr>
<td>SearchPanel</td>
<td>Selects the Search Tab and sets the search drop down to 'Name'</td>
</tr>
<tr>
<td>SearchRun</td>
<td>Searches for a product (fixed value currently)</td>
</tr>
<tr>
<td>HomePageAddItem</td>
<td>Randomly selects an item from the home page and generates the POST request to add item to the Cart. Checks that Item has been added.</td>
</tr>
</tbody>
</table>

Because of the way that state is maintained both on the server and on the client it is not possible to use Operations randomly as the state on the server corresponds to a certain state on the client. Therefore a FixedSequence (a faban harness term representing a specific order) of Operations was used which for the operational testing was as follows:

1. HomePage
2. SearchPanel
3. SearchRun
4. HomePageAddItem

At the next interaction, the Cart is empty due to the way the HomePage Operation is formed. It is possible to maintain the state when returning to the home page but this introduces other complexities when managing state and lies outside the scope of this paper.

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.

### Configuration

The following sections provide detailed information on both the hardware and software setup along with the specific software versions used. Additionally it includes guidance on how to most effectively utilize the system resources.

#### Software Configuration

The following software stack was used during the performance tests:
Software | Additional Information | Version
---|---|---
Oracle WebLogic Server | Application Server | 10.3.2
Oracle SOA Suite | WebLogic SOA implementation | 11.1.1.2.0
ADR | Application Developer Runtime | 11.1.1.2.0
JDeveloper | Integrated Developer Environment | 11.1.1.2.0 PS1
Fusion Order Demo | Demo Application | R1PS1
Oracle Database | Database | 11.2.0.1.0
Oracle Solaris OS | Operating System | 10 Update 8
JVM | Java Virtual Machine | HotSpot 1.6.0_18
Faban | Load Master / Harness | 1.0.1
Faban ADF Driver | Load Clients / Simulator | 1.0

**TABLE 2: SOFTWARE CONFIGURATION**

**Hardware Configuration**

The System Under Test (SUT) consisted of two systems. WebLogic Server 11g was deployed on Sun SPARC Enterprise T5240 and the Oracle 11g Database was deployed on Sun SPARC Enterprise M4000. The two systems were inter-connected via secondary 1Gb Network Interfaces to a private switch via a vLan configuration.

Load generation was managed via a Faban Master co-located with the Faban Agent running on a Sun Fire X2270 System. Other Sun Fire x2270s were available to act as Faban Agents as needed. The servers had specific hardware configurations as referenced in Table 3:

<table>
<thead>
<tr>
<th>Hardware</th>
<th>CPU</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun SPARC Enterprise T5240</td>
<td>SPARC-T2+ 8 cores @1.6GHz</td>
<td>128GB</td>
</tr>
<tr>
<td>Sun SPARC Enterprise M4000</td>
<td>4 * SPARC64-VII @2.6GHz</td>
<td>32GB</td>
</tr>
<tr>
<td>Sun Fire x2270</td>
<td>2 * Intel x5670 @2.93GHz</td>
<td>96GB</td>
</tr>
</tbody>
</table>

**TABLE 3: HARDWARE CONFIGURATION**
Tuning Recommendations

The following sections discuss tuning options and aspects to consider depending on your specific requirements.

Oracle Solaris Network

Oracle Solaris default network tuning parameters should be sufficient in most cases, however there are a number of TCP/IP tunables that can be useful for server workloads that respond to a large number of concurrent connections. For this testing the default TCP tunable tcp_time_wait_interval was reduced to 10000:

```
ndd -set /dev/tcp tcp_time_wait_interval 10000
```

Oracle RDBMS

In this test, the Oracle database instance also served as the Metadata repository for the application. This required the installation of the WebLogic SOA suite, although the SOA portion of the application (Composite Services) was not used for this testing. The database instance was setup as a default OLTP instance with the following additions as required by the Fusion Order Demo application:

- alter system set processes=500 scope=spfile;
- alter system set open_cursors=500 scope=spfile;
In order to act as a Metadata repository the Oracle Repository Creation Utility (RCU) utility needs to be run against the DB instance. Please refer to the link here for more details on RCU.

WebLogic Server

In order to fully support ADF applications, the WebLogic deployment requires that the following:

- Application Developer Runtime (ADR) component
- WebLogic SOA suite needs to be installed in order to support the use of an RDBMS as a Metadata repository.
- The WebLogic domain needs to have Oracle JRF installed from the domain configuration screen (or needs to be extended to add support for it)

The WebLogic domain should be configured for 'Production' mode. This is either done when the domain is being created or by setting the PRODUCTION_MODE variable to “true” in the WebLogic domain's setDomainEnv.sh file. Alternatively set Production mode for the domain in the WebLogic Console.

Oracle Java

All testing was performed with WebLogic server running on jdk1.6.0_18.

On Oracle Solaris running on SPARC-based processors, WebLogic Server can make use of a native performance pack. This is enabled by setting the LD_LIBRARY_PATH environment variable to point to

```
$MW_HOME/wlserver_10.3/server/native/solaris/sparc/libmuxer.so
```

before starting the instance (where $MW_HOME is the Oracle MiddleWare Home directory). To make this change permanent, add the following line to the domain's setDomainEnv.sh file:

```
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:
$WL_HOME/wlserver_10.3/server/native/sparc64
```

The performance pack gave a small performance improvement and was enabled for all tests other than the baseline tests. Results were always slightly better with the performance pack enabled.

Garbage Collection

JDK1.6 offers various options for configuring the Java Heap and for Garbage Collection. The default Garbage Collectors for the JDK1.6 Server VM are the Young Generation Parallel Throughput Collector (PS Scavenge) and the Parallel Tenured Generation Collector (PS MarkSweep). The number of threads used by these Parallel Collectors is configurable via a command line flag. The throughput
collector is designed to minimize the overhead of Garbage Collection through efficiency and does not necessarily concentrate on pause times. Long pause times can result in some users seeing very poor response times.

Using multiple threads in parallel to perform garbage collection on a generation is likely to reduce pause times, but there is a limit to how parallelized the collection of a generation can be made. The efficiency of garbage collection on a Java Heap can often be improved through consideration of the survival rate of objects allocated on the heap, in fact this was the driving force behind the development of generational garbage collectors. Many objects have very short life spans while some objects will live throughout the life time of the application. In between these extremes are objects whose life spans are likely to be linked to certain aspects of an application's behavior.

- Young Generation Collector
  The Young Generation has a collector that sacrifices accuracy for speed and is used to handle short lived objects as efficiently as possible. It is important to collect as much as possible within the Young Generation, and the factors that control this are the size of the Young Generation and the amount of time an object is allowed to 'live' in the Young Generation before being promoted to the Tenured Generation.

- Tenured Generation Collector
  The Tenured Generation has an 'accurate' collector, which is not as high performing as the Young Generation collector but which does collect every unreferenced object on the heap.

Which collectors to use, how big to make the generations and how long objects are allowed to live in the Young Generation before promotion are decisions that can often be made on the basis of well documented heuristics or else through detailed analysis of GC data. Other JVM options (and there are many) are often chosen as a result of the work of others on similar platforms and workloads.

Baseline testing
For all testing the following JVM options were used to gather GC data:

```
# WebLogic Server JVM GC stats
-XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+PrintGCDateStamps
-Xloggc:/tmp/gc.log
XX:+PrintTenuringDistribution was also used on some runs to analyze the lifecycle of objects within the Young Generation.

It was not possible to get any baselines for the default WebLogic JVM settings as the application would barely run under those conditions and would actually lock up under any kind of load. The WebLogic defaults are:

-server -Xms256m -Xmx512m -XX:MaxPermSize=128m
```
So the starting point was with a heap size of 3 GB and a Permanent Generation of 512 MB, using the 32-bit JVM as these settings were pushing the 32-bit java heap size limit

```
-server -Xms3g -Xmx3g -XX:PermSize=512m -XX:MaxPermSize=512m
```

The default collectors for the server VM are the Young Generation Parallel Scavenge collector (PS Scavenge) and the Tenured Generation Parallel MarkSweep collector (PS MarkSweep). Baseline settings were then taken with these values. Also the Performance pack was not enabled for the baseline tests. The domain was at all times running in Production mode.

![Baseline Throughput Graph](image)

**Figure 2: Baseline Throughput Graph**

As the baseline workload as shown in Figure 2, uses a Cycle Time of 10 seconds the maximum operations per second is always the number of concurrent users divided by 10.

The per operation 90th % Response Time averages in Figure 3 are interesting. They rise steadily and then decrease at 70 concurrent users.

![90th % Response Times](image)

**Figure 3: 90th % Response Times**

**ORACLE**
This begins to make sense when the per operation Max Response times (Figure 4) are looked at for the same runs.

This suggests a correlation between higher Max Response Times and lower 90th % Response Times. The high Max Response Times also made it difficult for the Faban test harness and load generator to maintain its Cycle Time targets and this may have been a factor in the anomalous behavior.

JVM Tuning

JVM Tuning was performed iteratively. For short runs, optimal throughput was obtained using the low pause garbage collector (ConcurrentMarkSweep collector), with the limitation that on long runs, the collector could not keep up with object allocation in the Tenured Generation and the occasional 'stop-the-world' full GC would be forced. These would often last for over a minute causing some operations to take > 100 seconds and thus skewing the overall results.

Overall the best performance (in terms of pause times and throughput over long runs) was obtained using the default Garbage Collectors and a 7 GB heap. This allowed us to obtain good throughput at 120 concurrent users. Larger heaps did not allow us to increase the number of concurrent users.

It was found during the testing that sizing the Permanent generation to 512 MB was adequate and seemed to avoid any unnecessary class unloading. However the defaults for the size of the Code Cache were marginally inadequate. The Code Cache is used to store compiled code that can be reused as needed by the Just-In-Time (JIT) compiler to avoid the overhead of recompiling bytecode that has already been compiled to native code. Most WebLogic workloads require between 10-15 minutes ramp up time in order to make optimal use of the JIT but in this case the Code Cache was filling thus forcing code to be pushed out of the Code Cache to make room for newly compiled bytecode. It's not documented as to what Cache policy the Code Cache uses but its likely to be LRU (Least Recently Used); As memory was not an issue on the SUT, the code cache was increased slightly to 64MB (with 8MB reserved) and this seemed to solve the issue (as viewed from graphs in the JConsole tool).

The optimal settings were found to be in the settings below, with results shown in Figure 5:
Tuning Oracle ADF applications on Sun T-series Servers

# WebLogic Server JVM tuning
-server -d64 -Xms7g -Xmx7g -XX:PermSize=512m -XX:MaxPermSize=512m
-XX:+AggressiveOpts -XX:CodeCacheMinimumFreeSpace=8m
-XX:ReservedCodeCacheSize=64m
-XX:CodeCacheMinimumFreeSpace=8m
-XX:ReservedCodeCacheSize=64m

Figure 5: Tuned Throughput
Conclusion

The application is heavy on object allocation and the expectation is that the heap size required to grow in proportionately with the number of users. At the moment a 7GB heap appears to be optimal for the maximum amount of concurrent users that a single WebLogic instance could service. Based on the data so far Solaris T-Series servers have performed well in handling the concurrent users. Since the performance tests are still underway a conclusive report on our findings will be published in the future.
References

- Oracle Application Development Framework:
- Oracle ADF FAQ: http://wiki.oracle.com/page/Oracle+ADF+FAQ
- Oracle Repository Creation Utility:
  http://download.oracle.com/docs/cd/E12839_01/doc.1111/e14259/rcu.htm
- Faban performance testing framework: http://faban.sunsource.net
- Oracle Sun SPARC Enterprise Servers: