ORACLE
FUSION MIDDLEWARE

An Oracle White Paper
April 2014

Tuning Oracle Traffic Director for Oracle Fusion Middleware, Business Applications – Technical White Paper

ORACLE

Executive Summary

This document provides technical information related to tuning Oracle Traffic Director and the underlying Linux system for achieving optimal performance while front-ending Oracle Fusion Middleware and Business Applications such as ATG Web Commerce, PeopleSoft, and E-Business Suite (EBS).

Introduction

Oracle Traffic Director (OTD) is an on-board, highly available Application Delivery Controller (ADC) to optimize application-to-application communication within Oracle's engineered systems such as Exalogic, and offers the following key benefits:

- High throughput, low latency HTTP(s), WebSocket, and TCP software load balancer.
- Rule-based request routing and reverse proxy server.
- Request rate limiting, throttling and QoS tuning.
- Flexible and fine-grained monitoring.
- High availability through standard based VRRP protocol.
- High performance SSL/TLS transactions including offloading SSL and HTTP compression.
- Built-in WebLogic optimizations such as cluster discovery, and connection persistence synchronization.
- Mod Security based Web Application Firewall to protect back-end applications from malicious attacks like CSRF, SQL Injection and so on.

Value Proposition

Customers can leverage Oracle Traffic Director – the application level controller within Oracle's engineered systems to:

- Maximize overall application throughput; optimize intra-application communication by leveraging the underlying InfiniBand as much as possible.
- Avoid Single Point of Failure (SPoF) throughout your deployment with OTD High Availability support for intra-application communication.
- Minimize application maintenance window by transparently draining incoming traffic for a given application server.
- Increase application availability to shape, limit, and throttle the incoming traffic.
- Monitor end-to-end deployment traffic through Oracle Enterprise Manager Cloud Control.
- Handle single sign-on, authentication and authorization deployment requirements through Oracle Access Manager WebGate plug-in.

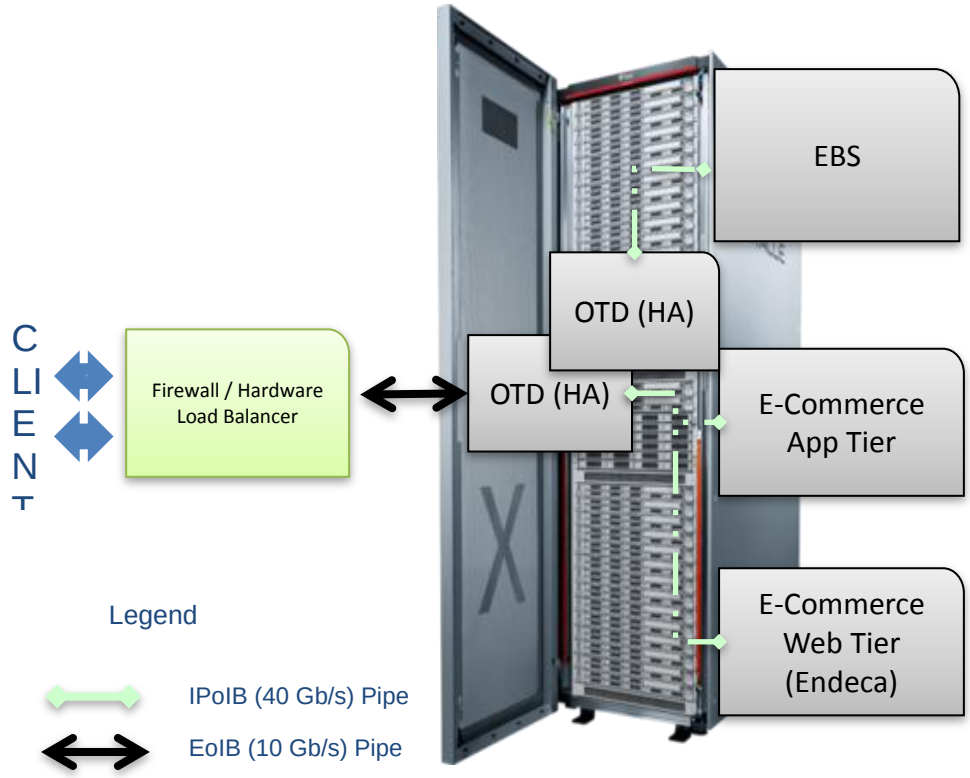

Figure 1 - OTD Value Proposition as Internal Load Balancer

Hardware Sizing

The overall application throughput and inherent OTD performance largely depends on the underlying application characteristics. And you should size hardware requirements based on your internal application performance testing. However, you can use the following information to help with hardware sizing:

- **CPU Sizing:** OTD on Exalogic (Virtual) with 2 vCPU (equivalent to 1-x86 processor core) load balances incoming requests to handle approximately 10,000 HTTP and 4,500 HTTPS transactions per second per core for an 8 KB HTTP payload with 1-2 millisecond response time window.
    - Allocate additional processors (as 2 vCPU increments) to OTD VM to handle higher throughput rate.
    - Overall system performance can drop by additional 5% if you have between 90-100% CPU utilization within the overall compute node.
- **Memory Sizing:** For optimal performance, allocate at least 8 GB RAM to OTD VM when provisioned with 2-vCPU instances. You should plan to increase additional memory with additional at 4 GB / 1-x86 processor core.

- OTD is primarily an Input/Output (IO) bound application, and to some extent a CPU bound application. So OTD does not require significant memory footprint. OTD should not need more than 16 GB RAM even with 4 or 8 vCPUs, unless you have explicitly configured to cache large amount of static content.
- If your application has static content pages such as JavaScript, Image files, HTML files and so on, then you can configure OTD to cache this content and serve it from in-memory. This scenario minimizes the overhead of frequently accessing these static pages from your web or application server. In this scenario, you should allocate additional 8 GB memory to handle this use case.

System Tuning

Oracle Traffic Director - as a software load balancer – is dependent on the overall system configuration and can perform optimally only when the system is appropriately tuned to handle the workload. The following key system level tuning parameters define OTD performance for high throughput applications (Example: Oracle ATG Web Commerce and WebLogic Java EE web applications):

- Sizing System File Descriptors
- Sizing Ephemeral Ports
- Tuning Linux TCP settings

System File Descriptors

OTD depends on the operating system resource such as *file descriptors* to accept incoming client connections, and to load balance requests from these connections to one or more origin servers. In addition, OTD auto tunes its various sub-systems such as Keep Alive connections, Connection Queue Size, Origin Server Connection Pool and so on based on the maximum number of file descriptors in the operating system. Hence, you should take sufficient care in tuning this critical system resource. For example:

- Administrators can restrict the number of file descriptors that every Linux user / process can consume using the system *'ulimit –n'* value. At the same time, OTD administrators need to ensure that OTD server instance process has enough file descriptor to meet the overall throughput requirement.
    - Users can tune the system level 'ulimit' limit by creating the following file within /etc/security/limits.d/:

```
  $ oracle > su –
  # root > cat > /etc/security/limits.d/ulimit.conf
 <otd-instance-user>  hard  nofile  <half-of-system-file-descriptor-limit-for-otd eg. 2097152 or unlimited>
 <otd-instance-user>  soft  nofile  <half-of-system-file-descriptor-limit-for-otd eg. 2097152 or unlimited>
```

```
<PRESS-CTRL-D-to-save-file>
```

- If this operating system resource is too low, then the Linux system and OTD will be unable to perform optimally under heavy load.
- If this operating system resource is too high, then OTD can appropriately auto tune and start with a large memory footprint.

To find out the current system limit for file descriptors, run the following command:

$ cat /proc/sys/fs/file-max

XXX -> This number provides the maximum file descriptor limit in this system.

To find out how many of the available file descriptors are being currently used, run the following command:

$ cat /proc/sys/fs/file-nr

XXX -> This number provides the file descriptors currently used by all applications in this system.

To handle heavy throughput workload (approximately 10,000 HTTP transactions per second per core), with incoming traffic from at least 30,000 concurrent users, here are some suggested defaults to guide you with sizing effort:

Figure 2 - Guidance on System Wide File Descriptors

OTD Configuration/System	Max File Descriptors	Comments
Only OTD in a system: Exalogic (Virtual) with Oracle VM hosting only OTD HTTP Load Balancing Service.	At least 2097152 Optimal Value = **6291456** '*ulimit –n*' value can be set to unlimited	Exalogic (Echo) Base Image template default value is greater than this. So, if you are using OTD on Exalogic (Echo) Base Image then do not make any changes. If you specifically increase Max. HTTP request processing threads within OTD, then you should correspondingly increase the overall system max file descriptor as well.
Only OTD in a system: Exalogic (Virtual) with Oracle VM hosting only OTD HTTP and TCP Load Balancing Service on 2vCPU	At least 2097152* 2 = 4194304 Optimal Value = **6291456** '*ulimit –n*' value can be set to unlimited	Exalogic (Echo) Base Image template default value is greater than this. So, if you are using OTD on Exalogic (Echo) Base Image then do not make any changes.
OTD along with other server applications such as WebLogic on Exalogic	At least 2097152* 2 = 4194304 Optimal Value = **6291456**	Exalogic (Echo) Base Image template default value is greater than this. So, if you are using OTD on Exalogic (Echo) Base Image then do not make any changes.

	Set appropriate limit for '*ulimit –n*' value	

To change the number of file descriptors in Linux, do the following as the **root** user:

1. Edit (or add) the following line in the /etc/sysctl.conf file:

```
fs.file-max = 2097152
```

value is the new file descriptor limit that you want to set.

2. Apply the change by running the following command:

```
/sbin/sysctl -p
```

For more information, refer to Tuning File Descriptor section within the Tuning Oracle Traffic Director for Performance section in the *Oracle Traffic Director Administrator's Guide*.

Ephemeral Ports

Linux operating system manages another pool of *file descriptors* commonly referred as *ephemeral ports*. These file descriptors can be described as short-lived transport ports for TCP/IP communications.

When OTD initiates a client TCP/IP connection to the back-end server such as WebLogic Server, then the Operating System (OS) temporarily allocates an *ephemeral port* - from a pre-defined range - to OTD, and this *port* is only valid for the duration of the communication session. At the end of the communication session, the OS reclaims this port – typically after some wait time commonly referred as '*fin_timeout*'.

If OTD either front-ends large number of back-end servers or frequently closes large number of back-end connections, then it is possible to exhaust the ephemeral port supply. In such situations, customers should adjust their TCP/IP ephemeral port range parameters to provide enough ephemeral ports for the anticipated server workload. In addition, you should:

- Ensure that the lower range is set to at least 9000 or higher, to avoid *Well Known* ports, and to avoid ports in the Registered Ports range commonly used by Oracle and other server ports.
- Set the port range high enough to avoid reserved ports for any applications you may intend to use.
- If you are using a separate Virtual machine (VM) for load balancing responsibilities (OTD VM), then you can lower the lower range to as low as '1024'.
- Linux systems have a typical 60-second timeout before an ephemeral port that is in TIME_WAIT state is reclaimed for reuse. On heavily loaded, high-throughput systems, you should reduce this timeout to 30-seconds so that this ephemeral port can quickly become available again for other connections.

With IPv4, customers can use the following command to check the current range for ephemeral ports:

```
$ cat /proc/sys/net/ipv4/ip_local_port_range
        32768 61000
```

Here, the lowest port (32768) and the highest port (61000) are set to the default range.

To change the ephemeral port range in Linux, do the following as the **root** user:

1. Edit (or add) the following line in the /etc/sysctl.conf file:

```
net.ipv4.ip_local_port_range = 9000 65535
net.ipv4.tcp_fin_timeout = 30
```

On OTD VM – where only OTD is running in the entire Linux system – you can alternatively use

```
net.ipv4.ip_local_port_range = 1024 65535
```

2. Apply the change by running the following command:

```
/sbin/sysctl -p
```

Tuning Oracle Traffic Director

Oracle Traffic Director performance is largely dependent on its following resources:

- Request Processing (aka Worker) Threads
- Connection Queue, Keep Alive Thread
- Persistent (Keep Alive) Connection with Origin Servers
- JVM Heap

This section focuses on tuning these key resources.

Recommendation: OTD 11.1.1.7 can auto-tune its key resources such as *Worker Threads*, *Keep Alive Threads, Connection Queues* and so on based on hardware resources such as *CPU*, and system resources such as *maximum OS file descriptors* available on the system. Hence, OTD product team recommends appropriately sizing the hardware and tuning the system, as described in the previous section, to handle a high throughput workload before tuning any of OTD's key tunable resources.

Monitoring OTD Performance

Monitoring OTD performance, during peak workload, allows you to identify the key resources to tune within OTD.

You can monitor OTD through:

- Oracle Enterprise Manage Cloud Control. For more information, refer to EM Cloud Control product documentation on how to configure and monitor Oracle Traffic Director through EM Cloud Control.
- Collect and monitor OTD data either through OTD Administration Command Line Interface (CLI) or through OTD Server Instance browser interface as mentioned within OTD product documentation. In addition, refer to the section below to collect OTD monitoring statistics through the OTD Administration CLI:

Collecting OTD Monitoring Statistics

You can use OTD CLI to collect OTD monitoring statistics on your staging as well as production systems. However, this interface provides the snapshot of OTD internal parameters at a given instant. Hence, you should collect OTD performance data for at least 10 to 15 minutes at regular interval (say every 30 seconds) to accurately identify the key OTD internal configuration parameters that affects OTD performance.

Here are the steps for collecting OTD performance statistics:

1. Create a file within the HOME directory of the OTD server instance runtime user.
    - For instance, if you have configured OTD Server Instance (<OTD_INSTANCE_HOME>/net-<config>) to run as 'oracle' user, then you will create the following file within the $HOME directory of the 'oracle' user.

$> cat > ~/.tadmrc
set tadm_user <OTD_ADMIN_USER>
set tadm_password <OTD_ADMIN_PASSWORD>
set tadm_port <OTD_ADMIN_SERVER_PORT – default is 8989>

<Press **CTRL-D** to save the above file.>

$> chmod 400 ~/.pw

2. Create a file, as shown below, within the ZFS shared storage so that it is accessible through all the machines hosting OTD Server instances.
    - You will need to provide '+x' permission to this file (chmod 755 <OTD_INSTALL_HOME>/otd-collect-data.sh) so that you can run this file either as a standalone script or through a '*cronjob*'.
    - You can run this script to collect OTD performance statistics. This script logs the statistics data to a file.

$> cat > <OTD_INSTALL_HOME>/otd-collect-data.sh

$> <OTD_INSTALL_HOME>/bin/tadm get-perfdump --config=<OTD_CONFIG> --node=<OTD_ADMIN_NODE_HOSTNAME> >> <OTD_INSTANCE_HOME>/net-config/otd-<HOSTNAME>-perfdata.log

$> <OTD_INSTALL_HOME>/bin/tadm get-stats-xml --config=<OTD_CONFIG> --node=<OTD_ADMIN_NODE_HOSTNAME> >> <OTD_INSTANCE_HOME>/net-config/otd-<HOSTNAME>-perfdata.log

<Press **CTRL-D** to save the above file.>

$> *chmod 755 <OTD_INSTALL_HOME>/otd-collect-data.sh*

Optional:

1. Leverage Linux 'cronjobs' tool to run this above script as a 'cron' job. This script will collect OTD performance statistics within <OTD_INSTANCE_HOME>/net-config/otd-<HOSTNAME>-perfdata.log.
2. Leverage Linux 'logrotate' tool to compress and rotate the above collected monitoring information from each system.
    - For example, a 'logrotate' configuration file like below compresses and rotates OTD monitoring statistics
    - Change to 'root' and create a file using the following command:
      $# cat > /etc/logrotate.d/otd-instance-monitoring

```
<OTD_INSTANCE_HOME>/net-<config>/otd-<HOSTNAME>-perfdata.log {

  missingok

  notifempty

  compress

  daily

  rotate 5

}
```

Tuning HTTP Request Processing (Worker) Threads

Oracle Traffic Director product documentation provides in-depth background on this key resource and how it uses this resource to handle the workload within the section Tuning the Thread Pool and Connection Queue. In a nutshell, Oracle Traffic Director assigns an individual thread from its HTTP thread pool to process a given HTTP request. This HTTP thread becomes available to process another HTTP request once it completes processing the existing HTTP request. A typical response time for an HTTP request is anywhere between few microseconds to few milliseconds. So a few hundred OTD HTTP request-processing threads can process several thousand HTTP requests per second.

In Exalogic (Virtual), typical sizing for OTD VM is either 2-vCPU or 4-vCPU. Hence, OTD 11.1.1.7 defaults to creating a maximum of either 512 threads (when running on a 2 vCPU system) or 1024 threads

(when running on more than 2 vCPU systems). This configuration allows OTD to process 10,000 HTTP requests per second per core provided the response time for a HTTP request is few milliseconds. However, Oracle business applications such as SOA, Forms, ATG Web Commerce, and PeopleSoft have response times that are greater than few milliseconds. Here you should increase OTD's maximum HTTP request processing threads to ensure OTD has enough horsepower to handle the incoming traffic with slower response time. In addition, if you find that OTD and the back-end applications are not saturating the underlying CPU, then you should increase OTD maximum request processing threads to a higher value as well.

Here are some thumb rules to help you decide when you should increase OTD maximum HTTP request processing threads:

- Monitor OTD performance and specifically look for request processing threads information as described within section - Reviewing Thread Pool Metrics product documentation and Monitoring OTD Performance section.
    - If you observe that the total number of request-processing threads created is consistently near the maximum number of threads, consider increasing the thread limit. Otherwise, requests might have to wait longer in the connection queue; and if the connection queue becomes full, further requests are not accepted.
    - If the average queuing delay (see Connection Queue Metrics) is significantly high in proportion to the average response time, then this is also an indication that you should increase the maximum HTTP request processing threads.
- If OTD is hosted on 8 vCPU VM, then you might also want to increase Maximum HTTP request processing threads to a higher number such as 8192 to ensure that OTD has enough horsepower to utilize the underlying CPU resources.
- If OTD is front-ending large number of back-end (origin) servers such as WLS or ATG Endeca servers with more than sub-second response times, then you should consider increasing the maximum HTTP request processing threads to a higher value.
    - For Example, if your web application web page response time is in sub-seconds, then you should increase OTD maximum request processing threads to handle large number of incoming requests.
    - When OTD is front-ending requests to Endeca servers, where typical response time is slower, then it is not uncommon to have maximum OTD request processing threads as 16,384.
- Again, you should continue to monitor OTD performance to ensure that OTD has enough horsepower left to handle any sudden burst of incoming workload. For more information, see section Reviewing Thread Pool Metrics and Connection Queue Metrics.

Note: If you need to increase 'Maximum HTTP Request Processing Threads' within OTD, you will need to ensure that your underlying system has enough system file descriptors. For example, if you increase

'Maximum HTTP Request Processing Threads' to 4096, then you should ensure that your overall system file descriptor is configured to be at least **6291456**. (For more information, see section System File Descriptors.)

Recommendation: Oracle Traffic Director product team recommends increasing only the 'Maximum Threads' value (and leaving the Minimum Threads to default value) as shown here:

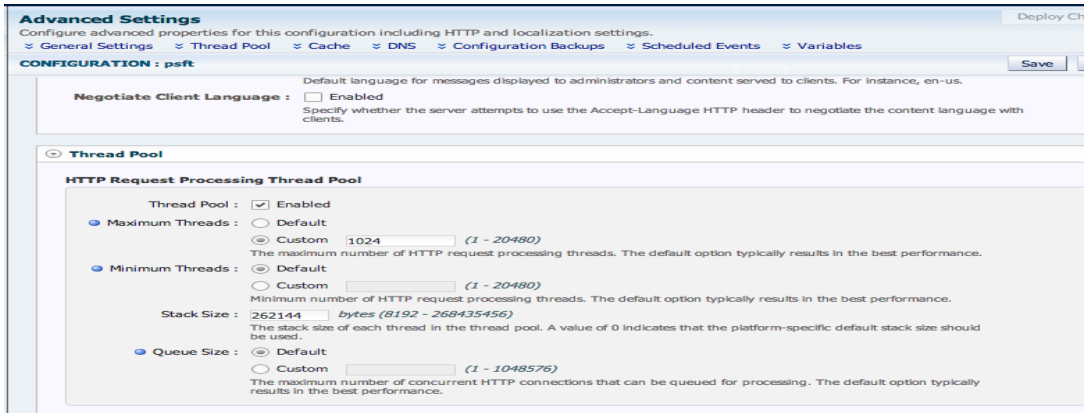

Figure 3 - Increase Max. HTTP Processing Threads

Tuning Connections, Keep Alive / Acceptor Threads

OTD has the ability to appropriately auto tune these values based on the overall number of maximum number of OS file descriptors. So Oracle Traffic Director product team recommends that this value be left to 'Default' for most scenarios.

Oracle Traffic Director product documentation provides in-depth background on this key resource and how you can leverage the collected OTD performance data to appropriately tune these key resources. For more information, see section Reviewing ConnectionQ metrics.

Tuning Persistent (Keep Alive) Connection with Origin Servers

Oracle Traffic Director, by default, does not maintain persistent connections with back-end servers while handling non-idempotent requests such as POST requests. This behavior ensures that OTD is compliant with HTTP/1.1 specification. However, this spec compliance comes at a huge performance cost to the overall throughput when significant number of incoming requests contains POST requests. This is especially true with use cases such as Forms, SOA, PeopleSoft, and ATG Web Commerce applications where HTTP POST requests are very common.

If your back-end application can handle non-idempotent requests (such as HTTP POST requests) as a persistent connection – commonly seen within Oracle Fusion Middleware and Business Applications such as SOA, PeopleSoft, ATG Web Commerce and so on – then you can configure Traffic Director (OTD) to always maintain persistent connections as shown here:

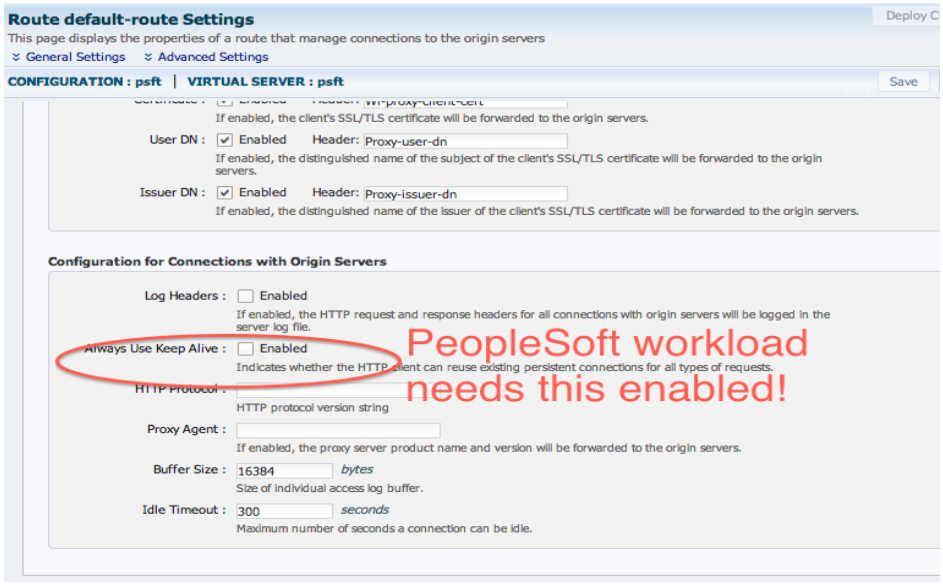

Figure 4 - Maintaining Persistent Connections for POST requests

For more information, refer to Tuning Connections to Origin Servers section within Oracle Traffic Director product documentation.

For use cases such as SOA, PeopleSoft deployments, response to a given request is more than 30 seconds. In this case, you will need to increase the Keep Alive timeout with the origin server so that OTD can attempt to maintain persistent connections with these servers as much as possible. In this case, you can increase the Keep Alive timeout to a higher number (say at least 60-90 seconds) depending on how long it takes for back-end server to process the request. You can increase this value under Virtual Server (VS that processes this request) -> Route (Route that sends incoming traffic to this back-end server) as shown below:

Figure 5 - Increase Keep Alive Timeout for maintaining Persistent connections

JVM Heap

Oracle Traffic Director product includes two key components:

- Java based Administration module (Administration Server or Administration Node)
    - This component is responsible for providing administration interface, managing server configuration files, and also for server instance lifecycle on one or more machines.
    - This component bundles a default JVM heap size set to 128 Mb. Oracle Traffic director product team recommends that this heap size is kept at default and is **NOT** increased to a higher value.
    - This component resides within <OTD_INSTANCE_HOME>/admin-server directory and can be safely shut down without affecting load balancing service.
- Server Instance
    - This component is responsible for providing actual load balancing service. This component does NOT use Java.

In nutshell, you should not tune any JVM heap size or do not need any JVM tuning expertise to tune OTD.

Annexure - Tuning Checklist

OTD front-ending Oracle SOA, PeopleSoft, and Forms deployment:

- Ensure underlying system is tuned to handle the workload.
    - Check for at least 65k file descriptors to be available to OTD. You can get this information by running 'ulimit –n'. Otherwise, you will need to increase the overall system file descriptors as mentioned in section System File Descriptors.
- Ensure OTD maintains persistent connection with the client.
    - Increase OTD KeepAlive Timeout with the client if you notice a lot of Keep Alive timeouts within OTD performance monitoring data.
    - Update Keep Alive Timeout under OTD Configuration -> Advanced Settings -> HTTP-> Keep Alive) . For more information, see section **Error! Reference source not found.**.
- Ensure OTD maintains persistent connection with the back-end (origin) server.
    - Update OTD Routing Policy to allocate higher Keep Alive Timeout value depending on how long you expect a response to complete for a given request in peak load (Virtual Server -> Routes -> General Settings -> Keep Alive Timeout).

- Update Routing Policies (Virtual Server -> Routes -> Advanced Settings -> Configuration for connection with Origin Servers) and enable 'Always Use Keep Alive' settings.

OTD front-ending Oracle Commerce (ATG) Web Applications

- ATG applications typically tend to require high throughput with above average response times.
- Provision vCPU for OTD based on the 'Hardware Sizing' section especially for Exalogic (Virtual) to meet overall application throughput requirements.
- Ensure underlying system is tuned to handle the workload.
    - Check for at least 1M file descriptors to be available to OTD. You can get this information by running 'ulimit –n'. Otherwise, you will need to increase the overall system file descriptors as mentioned in section System File Descriptors.
    - If OTD front ends large number of back-end servers (say > 4) while handling ATG deployment, then ensure that the system has enough ephemeral ports available. For more information, see section - Ephemeral Ports.
- Ensure OTD maintains persistent connection with the back-end (origin) server.
    - Update Routing Policies (Virtual Server -> Routes -> Advanced Settings -> Configuration for connection with Origin Servers) and enable 'Always Use Keep Alive' settings.
- Ensure OTD has enough HTTP Request processing Threads to handle the workload. For more information, see sectionTuning HTTP Request Processing (Worker) Threads.
    - Note that for a high throughput site, users can start with the following values as default and tune them based on their load testing exercise.
    - Default OTD maximum HTTP Request processing threads, while front-ending ATG web applications, can be around 4096 threads.
    - Default OTD maximum HTTP Request processing threads, while front-ending Endeca applications, can be around 16384 threads.

Note: Oracle recommends our customers to load-test their environment and measure their performance, including in OTD (as mentioned in section Monitoring OTD Performance), and increase OTD maximum request processing threads based on this investigation.

Conclusion

Oracle Traffic Director is a high performance software load balancer for Oracle Fusion Middleware and Business application for Oracle engineered systems. As with any software applications, Oracle recommends

that our customers deploy, measure, and appropriately tune the software load balancer to meet the overall throughput requirements.

White Paper Title
[Month] 2013
Author: [OPTIONAL]
Contributing Authors: [OPTIONAL]

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

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200

oracle.com

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

Hardware and Software, Engineered to Work Together