Oracle® Transportation Management

Application Scalability Guide

Release 6.5.3

Part No. F82869-01

September 2023

ORACLE

Copyright Notice

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Contents

Figures

Tables

Send Us Your Comments

Oracle® Transportation Management Application Scalability Guide, Release 6.5.3

Part No. F82869-01

Oracle welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the title and part number of the documentation and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: otm-doc_us@oracle.com

If you would like a reply, please give your name, address, telephone number, and electronic mail address (optional).

If you have problems with the software, contact Support at https://support.oracle.com or find the Support phone number for your region at http://www.oracle.com/support/contact.html.

Preface

Intended Audience

The Application Scalability Guide is intended for clients, system administrators, and consultants.

> **Note**: This version of the document assumes you are on the latest release of 6.4.3 and have all of the critical and recommended Scalability patch fixes.

> **Note**: All diagnostic servlets referenced in this document are provided as-is.

Related Documents

- Oracle Transportation Management Administration Guide
- Oracle Transportation Management Installation Guide
- Oracle Transportation Management Technical Architecture Guide
- Oracle Transportation Management Integration Guide

Definition of Related Scalability Terms and Concepts

- **Machine**: A computer hosting one or more instances of Oracle Transportation Management.
- **Web Instance**: An Oracle HTTP Server service, providing fixed Oracle Transportation Management content to browsers
- **Application Server**: A WebLogic server running Oracle Transportation Management. This includes a servlet container to provide user screens to the web instance and a J2EE container to process business functionality.
- **Cluster**: A logical grouping of application servers dedicated to a set of work.
- **Topology**: A specification of all web instances and application servers, along with the current distribution of work among them.
- **Scalability**: Use of multiple web instances, application servers, and clusters to increase Oracle Transportation Management throughput by adding machine resources.

Definition of Related Java Terms and Concepts

- **JMS (Java Message Service)**: A Java message-oriented middleware API for sending messages between two or more clients.
- **RMI (Remote Message Invocation)**: A Java protocol built over TCP/IP, allowing one Java process to call methods in another.
- **JDBC (Java Database Connectivity)**: Industry standard for database-independent connectivity between Java and a range of databases.

1. Introduction

Scalability is the name given to Oracle Transportation Management's proprietary solution of application server clustering.

Why Scalability?

By running Oracle Transportation Management on multiple web and application servers, Scalability provides:

- **High Availability**. Used in tandem with web-tier load balancers, Scalability can ensure system availability after server failure. User requests and background work are automatically redirected to available servers when a hardware failure occurs or when an application server becomes unresponsive.[1]
- **Horizontal Scaling**. As demand increases, Scalability can quickly adapt to increased work, distributing work across a growing network of servers.
- **Support for Dedicated Resources**. Scalability allows resource-intensive background work to be dedicated to a group of servers. This can improve the throughput for user activity as users are not vying for the same server resources as background processes. Note that the use of Dedicated Resources is supported only in On Premise environments.

Scalability Tradeoffs and Restrictions

The addition of multiple web and application servers allows for increased availability and throughput but incurs some overhead. Data cached in the application-tier must be duplicated across all application servers. Access to critical business or system objects must be synchronized across all application servers.

High Availability

Work processed by Oracle Transportation Management servers comes from two primary sources: user requests and polled work. When a user logs into the system, the web instance establishes a user session and all future requests from the user are sent to the web instance managing its session. Typically, these requests are ad-hoc and of short duration. Longer running, resource intensive work (e.g. recurring bulkplans, inbound integrations) are queued up in database request tables available to any application server.

To provide high availability for user requests, Oracle Transportation Management requires the use of a Layer 7 load balancer. This allows for HTTP health checks that automatically detect failed hardware or unresponsive software. On failure, the load balancer automatically redirects users to a healthy web and application server.

High availability for polled work is an implicit capability of Scalability. Each application server periodically polls the database for long-running work. If one server fails, the other servers pick up the load during their normal polling cycles.

Note that High Availability is not a hot failover solution. All servers participating in a Scalability environment are actively processing user and background work. This avoids underutilizing hardware resources for a rare failure event.

---

[1] Due to user session loss, users may need to relogin when redirected by the load balancer to another web instance.

Horizontal Scaling

A major issue with any software is performance. Scalability can help increase performance by providing application server horizontal scaling.

Horizontal scaling can address many performance problems, but at a cost of data duplication and increased synchronization between servers. A rule of thumb is adding an application server may increase throughput by up to 80%. Adding more application servers reduces this best-case throughput, as servers spend more time in contention and notification functions needed for data integrity.

Before implementing Scalability to enhance performance, it's important to determine if the application server is the bottleneck of Oracle Transportation Management. If the application server memory is exhausted, threads are maxed out, and/or the CPU of the server running the application server is pegged, then Scalability can add additional resources to the application tier and improve overall performance. Scalability will be less useful if the database or the web instance is the performance bottleneck. Proper tuning of the Oracle Transportation Management software, the application server, the JVM and the database should be explored first before moving to a Scalability environment. Scalability could add unnecessary processing, and compound the overall performance.

Database Scaling

Oracle Transportation Management fully supports the clustering of database work using Oracle RAC Technology. Implementing a RAC solution may often improve performance more than clustering application servers via Scalability. For more information on minimum server requirements and specifications for a RAC implementation, please consult the Administration Guide and the Technical Architecture Guide. Details regarding implementing a RAC solution are not covered in this document.

2. What Is Scalability?

Scalability is the name given to Oracle Transportation Management's proprietary solution of application server clustering. Scalability allows for multiple application servers to run in separate processes on the same server or on separate physical servers. The application servers then communicate with each other to ensure data integrity and synchronization of data caches, supporting horizontal scaling.

Scalability is a custom-written mechanism to handle application server clustering and distributed communications for the Oracle Transportation Management application. It does not use the WebLogic clustering solution. Scalability is a propriety scaling solution only for Oracle Transportation Management.

Base Architecture

To understand how Oracle Transportation Management and Scalability fit together, the basic Oracle Transportation Management architecture should be reviewed. The Oracle Transportation Management architecture is a three-tier Java web application, consisting of a web tier, an application tier, and a database tier. The database tier is an Oracle Database server. The web tier consists of the Oracle HTTP Server (OHS) serving static content and acting as a proxy for all dynamic content. The application tier is an Oracle WebLogic Application Server consisting of two logical components: a User Interface (UI) Component servicing servlet requests from the web-tier, and a Business Component handling core transportation functionality.

User requests are sent by a browser to an OHS web instance. While the web instance can return static content directly to the browser, the vast majority of requests in Oracle Transportation Management are servlet requests that are redirected to the UI Component on the application server. OHS passes these requests to the application server using the OHS **mod_wl_ohs** connector.

All servlet requests coming into the application server are handled by the WebLogic servlet container. UI Component code integrated with the servlet container passes business requests to the Business Component. While in previous releases the communication between the UI Component and the Business Component leveraged J2EE[2], as of release 6.4.3 this is no longer necessary. The UI Component and Business Component are co-located in the same WebLogic server, allowing for communication to be via direct Java calls. This avoids any overhead due to data marshalling and transport.

All database queries and updates are performed by the Business Component, using JDBC and SQLNet to communicate with a database server. Results from the database tier are returned to the Business Component and then to the UI Component. The UI Component transforms data into HTML content for the web instance, which then returns the HTML to the browser.

Daemon threads in the Business Component periodically query the database for scheduled work and inbound integrations. Such tasks spawn off complex workflows that typically represent the bulk of transportation work performed.

[2] This was necessary since the UI Component ran in a separate WebLogic instance from the Business Component.

Scalability Architecture

Figure 2-1 displays the architecture in a Scalability implementation. Requests from browsers are sent to a third party load balancer[3], which in turn dispatches the requests to individual Oracle Transportation Management instances through an OHS web instance. When the web instance receives a servlet request, it routes it to the corresponding WebLogic server running in its Oracle Transportation Management instance. WebLogic's servlet engine invokes UI Component logic and passes any transportation work to the Business Component. The Business Component interacts with data to service the request.

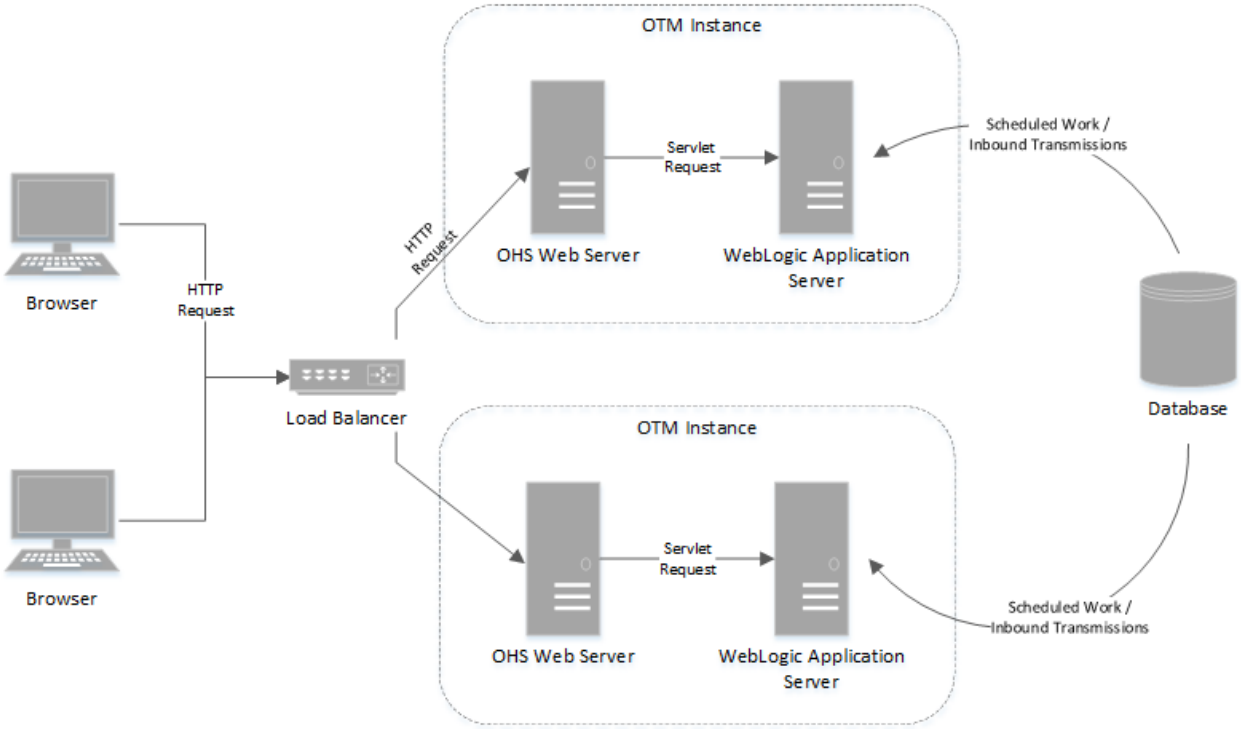


Figure 2-1: Scalability Architecture

The Scalability architecture consists of four separate components: user load balancing, background work polling, data synchronization, and cross-process coordination.

User Load Balancing

The distribution of user requests to separate web instances is the sole responsibility of the load balancer. Note that On Premise customers must provide their own load balancer while Cloud customers use balancers provided by Oracle Cloud.

Since the web instance maintains the state information for each user in a user session, Oracle Transportation Management requires the use of either sticky sessions by the load balancer. Sticky sessions allow multiple user sessions to be balanced across web instances, but delegate all work for a particular user session to the web instance first delegated user work (i.e. typically user login). If the web instance servicing a user session goes down and a load balancer redirects the user to another web instance, the user's session information is lost and they must log in again. This occurs because of

[3] This component is not provided by Oracle Transportation Management.

the loss of session between the web instances. The sessions are "sticky" to the web instance the user was first logged into, and the sessions are not replicated to other web instancess. So, in a non-single sign on environment, the user would receive the login page when they lose their session. But in a single sign on environment, they would get exceptions because the session does not exist.

Load balancers are critical for a High Availability environment. If a web or application server goes down or becomes unresponsive, it is the responsibility of the load balancer to detect it. The balancer should then redirect the user to a new or replicated session on a healthy web and application server. To meet this requirement, Scalability requires the use of a Layer 7 web balancer, implementing periodic HTTP health checks against the web instance.

The configuration of a load balancer, including specific instructions to configure HTTP health checks is dependent on the specific load balancer selected. It is beyond the scope of this document.

Background Work Polling

The distribution of background work to application servers is achieved by staging this work into database tables. Daemon threads running on each application server, poll these tables and pull of work as needed.

The following types of work are distributed by background polling:

- **Scheduled Process Control requests**. Application servers in a cluster poll the Process Control Request table to retrieve scheduled or recurring requests that are ready for processing. This can include work such as recurring bulkplans or data purge. Note, though, only asynchronous work can be scaled across application servers. A user who opts to execute a Process Control workflow synchronously (i.e. Now) forces that workflow to run on the application server servicing his user request.
- **Data Queues**. Applications servers in a cluster also poll the Data Queue tables to retrieve queue requests ready for processing. In particular, Inbound Integrations are automatically distributed across servers if they are staged in the Integration In data queue. For a scalable environment, this is the recommended method for staging inbound transmissions for processing. See the Integration Guide for more information on setting up a data queue for this purpose.

The threads that contend for background work have two mechanisms to ensure balanced load across servers. For work identified as Long Running Tasks, a server only retrieves the work if a thread is available to process it. If not, another server can retrieve the work when it next polls the table. E.g., by default Bulkplans run on a **batch** workflow thread. If an application server has no idle batch threads, it will ignore any pending Bulkplans when retrieving work from the Process Control table. Only a server with an idle **batch** thread will process the Bulkplan. Please refer to online help for information on configuring and monitoring Long Running Tasks.

Even when workload is light, scheduled processes are still distributed across servers over time. This is ensured by avoiding fixed polling intervals for each server. Instead, a server polls work tables every minute ± 15 seconds, where the actual polling time is distributed randomly between 45 and 75 seconds. This avoids server bias for processes scheduled at fixed clock times.

Data Synchronization

Data synchronization uses server messaging to keep data synchronized across application server caches. At a very high level, data synchronization works by having application servers communicate with each other using the Java Message Service (JMS). JMS allows application servers to write, send, receive, and read messages. Two or more application servers running Oracle Transportation Management send messages to each other and receive messages from each other, so they can communicate and coordinate what processes need to be done. These messages vary based on what business process needs to run, whether certain bean or object caches should be flushed or updated, or

whether it is a message for a network topology change. Every application server in a Scalability environment acts as a JMS server and client.

Cross Process Coordination

Scalability also uses a business locking component to achieve cross application server and JVM data integrity. Rather than rely on low-level record locks in the database, Scalability uses a proprietary locking mechanism to synchronize high-level business objects used in Oracle Transportation Management, such as shipments and orders. These are referred to as Object Locks. An object lock stops two processes from simultaneously modifying the same business objects, whether the processes are on a single application server or multiple servers.

Scalability Business Locking – In-Memory Locking vs. Database Locking

An in-memory lock is a local lock on an object achieved in the local JVM through a mutual exclusion mechanism built into the Java language. A non-scalability application server running Oracle Transportation Management uses only an in-memory lock to coordinate process access to business objects. Scalability uses in-memory locks locally, but also needs to perform locking across application servers and JVMs.

To achieve cross-machine locking, Scalability using two mechanisms: hard locks with the Oracle Database row locking mechanism, and soft locks using the application OBJECT_LOCK table. As both of these locks are points of contention in a Scalability implementation, it's important to understand how and when they are used.

Oracle Transportation Management Hard Locks – BNG Select for Update

The Oracle Transportation Management hard lock relies on and uses a SELECT FOR UPDATE statement to synchronize access to common data across application servers in a Scalability environment. This type of row locking is primarily used in the Oracle Transportation Management business number generator (BNG) to guarantee the same business number is not generated in two separate processes on two application servers.

Oracle Transportation Management Soft Locks – OBJECT_LOCK table

The Oracle Transportation Management soft locking is done in a Scalability environment by storing the lock record in the OBJECT_LOCK database table. After a process retrieves a local Java lock on the business object, it checks the OBJECT_LOCK table for a record to see if another application server machine is holding the lock. When releasing a lock, a Scalability-enabled application server will notify all other application server(s) by using a JMS message to reattempt the object lock.

The most important concept to understand is that most Oracle Transportation Management business object locks only apply to the top-level business object in a transaction such as a shipment or an order.

The number of OBJECT_LOCK records does accumulate over time, and these are now periodically purged by a staged daily cleanup process. See the Additional Recommendations section for more details about these records being removed.

Scalability Topology

The topology of a Scalability implementation is the collection of web instances, application servers, and clusters allocated to Oracle Transportation Management. It is important to understand the relationship between these resources and how they impact the distribution and processing of work.

What Is a Cluster?

An Oracle Transportation Management cluster is a grouping of application servers to handle some set of work. Each application server represents an installed instance of Oracle Transportation Management.

In the standard Scalability configuration, all application servers are assigned to a single DEFAULT cluster. This cluster polls all scheduled Process Control and Data Queue work.

Functional Clusters

In more advanced configurations, additional clusters may be set up to handle specific Process Control functions and Data Queues. This allows application servers to be dedicated to long-running background work, freeing up the DEFAULT cluster to handling user requests routed to it by the web load balancer. Setting up functional clusters should be approached with care. Note that:

- Dedicated clusters can lead to significant under-utilization of resources
- When a server in a dedicated cluster fails or becomes unresponsive, additional work is handled only by other servers in that cluster. If no other servers have been assigned to the cluster, critical work can back up indefinitely.
- Any work not explicitly assigned to a functional cluster, whether from Integration, Process Control or Data Queues, is automatically handled by the DEFAULT cluster.

Functional clusters are not available in Cloud environments.

Scalability Dynamic Topology

Scalability assumes that all web instances and application servers that could be active in a topology are specified in the glog.properties file for each installed instance. To add additional servers requires modification of these properties and cycling of all servers.

Cluster topology, though, can be changed while the system is running. Screens are available to allow for a dynamic configuration of:

- **Clusters**: Clusters can be added or removed.
- **Server Assignment:** Servers can be added to or removed from a cluster.
- **Work Assignment**: Work can be added to or removed from a cluster.

It is important, though, to maintain a DEFAULT cluster with at least two application server. This cluster handles any work not explicitly assigned to another cluster.

Changing the Scalability topology dynamically forces all of the application server machines and web instances to update their topology maps by broadcasting a JMS message to all servers.

3. Scalability Limitations

Performance Issues

Shared Resource Overhead

Enabling Oracle Transportation Management Scalability requires coordination of shared resources across all the application servers. This reduces performance since the coordination must be implemented via database objects rather than locally in the app-tier JVM.

In a Scalability configuration, there is an expected performance decrease, per server, compared to a single server configuration without Scalability. The typical performance overhead for a single application server is around 20%. While increased throughput is gained, maximum throughput for any single server is reduced.

JMS Overhead

In a Scalability environment, every data change may[4] be broadcast to every other server. Though the broadcast message is lightweight, the volume of JMS messages can reduce the performance of each application server.

Additional Object Contention

The high Scalability configuration may increase object contention across the servers. For example, order updates coming from upstream systems may be directed to disparate servers. If two of these updates occur on the same order, the servers need to synchronize access to the order. As additional servers increase the potential throughput of integrations, additional contention may occur as well. This may have a dampening effect on the expected performance improvement.

Data Synchronization

Inherently with JMS, there is the possibility of message latency. Oracle Transportation Management does not use persistent JMS because of additional performance concerns. However, an Oracle Transportation Management client can configure the application server so JMS persistence is enabled. There is a possibility of a delay for the JMS messages especially when there is a message for synchronization across application servers.

For example, it is possible that two active application servers could be performing a transaction on the same shipment. The first application server could finish its transaction, broadcast the required messages to the other application server, and by the time the other application server receives the messages it could have already committed the shipment. The shipment modifications done on the second application server could overwrite the modifications done on the first application server. Oracle Transportation Management does provide the ability to automatically detect this latency and fail the second transaction.

The latency of JMS messages could be minimized by increasing the dedicated JMS threads of the application server. This is an application server-specific configuration.

[4] A number of optimizations reduce the need for JMS messages based on knowledge of business object triggers within Oracle Transportation Management.

4. Changes from Earlier Versions

The Scalability architecture in Oracle Transportation Management 6.4.3 departs significantly from that in earlier releases. This is due to the continued evolution of OTM/GTM architecture.

Prior to release 6.4.3, the Apache web server was replaced with Oracle HTTP Server and Apache Tomcat servlet container with WebLogic. In Release 6.4.3, the WebLogic servlet container and the WebLogic J2EE container have been consolidated into a single layer called the application-tier. It is no longer possible to have dedicated servers for the web tier. The primary motivation for this change is that the OTM Application does not truly have a "web only" layer. All user activity in the user interface (search, view/edit, actions) requires significant interaction with the application layer. As the user interface technology has evolved, the line between application layer and web layer has blurred to the point that these layers are no longer significantly distinguishable. Large customers with multiple dedicated web servers will need to evaluate their current server configuration and adjust to this change. In many cases the current applications servers will be sufficient to handle the load of both user interface and business logic. In other cases, a marginal increase in the resources of the current application tier will be sufficient.

The original design of the OTM Scalability architecture was intended to deal with the memory constraints of 32bit JVMs, which often led to resource exhaustion in the WebLogic J2EE server holding the OTM Business Components. The legacy architecture provided substantial flexibility that was often misunderstood and misconfigured. In many cases customers implemented a complex configuration which led to an imbalanced workload and under-utilization of hardware resources. In Release 6.4.3, significant enhancements have been made to reduce the complexity of the configuration and to encourage better utilization of hardware resources. The legacy Scalability design was based on direct routing from the web server to the app server and between app servers. In most cases this led to an imbalanced workload and under-utilization of hardware resources. The new design favors a homogeneous environment for distributing short running tasks (i.e. UI activity, actions, integration, etc.) across all servers and a dynamic load balanced model for long running tasks (i.e. Bulkplan). If you are currently running with OTM Scalability enabled, please review this entire guide in detail to understand the impact.

Cluster Homogeneity

Prior to release 6.4.3, the web server routed a user request based on work content to an appropriate application server. This took into account any cluster dedicated to the work request, application server resource differences and any explicit failover clusters servicing a down application server.

With the consolidation of the UI and business components into a single WebLogic server, this routing is no longer available. Any user request coming into a web instance will be forwarded to its dedicated application server, regardless of content. This optimizes the processing of an individual request by avoiding RMI calls to a remote server. The UI and business components work together in the same JVM on the same memory.

All servers within a cluster are assumed to be identically resourced. The server weighting, available in earlier releases, has been removed.

Load Balancer Failover Detection

In previous releases, application server failover detection was the responsibility of the UI Component running in the web server. When an RMI call failed due to a down or unresponsive application server, the UI component would automatically redirect the request to an explicit failover cluster, or to another server in the target cluster if a failover cluster was not specified.

With release 6.4.3, both web and application failover detection is the responsibility of the load balancer[5]. Failover clusters are no longer available. To ensure both failed and unresponsive application clusters are properly detected, a Layer 7 load balancer should be used that supports custom HTTP health checks.

Scaling Scheduled Work and Inbound Integrations

Release 6.4.3 has improved the scaling of scheduled work (i.e. Process Control requests) and inbound integrations[6]:

- Work polling can now take into account server load. With the new Long Running Task feature, a server can restrict polled work to the number of available threads to process it. This improves work distribution across servers in a cluster and avoids workflow queue backup.
- Server bias has been removed. In prior releases, work scheduled for even clock intervals (e.g. at an exact hour) was picked up by the server whose polling was closest to the clock interval. In 6.4.3, the polling interval has been randomized to ensure all servers have equal chance of polling a particular work request.

Note that, in an On Premise environment, a cluster may still be dedicated to a particular set of scheduled work or to inbound integration. Load balancers can direct user activity to a default set of web and application servers, while another cluster of application servers can service long running tasks such as scheduled bulkplans. This advanced configuration should be used sparingly, as the base architectural improvements should minimize the need for such work separation.

[5] Not included as part of Oracle Transportation Management

[6] When queued into an Inbound Integration data queue

5. Configuration

To enable a Scalability implementation, On Premise customers must adjust the default Oracle Transportation Management configuration to account for multiple web and application servers.[7]

The following steps should be used to setup a single cluster with *n* servers where each server represents a distinct installation of Oracle Transportation Management software. Each installation will consist of an Oracle HTTP web instance (OHS) along with an Oracle WebLogic server dedicated to servicing it.

Server Identification

For Scalability, each web and application server must be uniquely identified. For each application server determine:

1. Its unique T3 URL, accessible by all web instances and other application servers. E.g. **t3://myApp01.domain.com:8001**, where **myApp01.domain.com** is the DNS entry for the application server and **8001** is the WebLogic report responding to RMI requests (the OTM_DOMAIN_LISTEN_PORT on installation).
2. Its unique display name. This should be an upper-case string no longer than 50 characters long. It will be used by the Oracle Transportation Manager user interface as an identifier for the server.[8]
3. Its unique host name. The installer will use this host name to construct a unique WebLogic target name of the form **gc3webapp-**<hostname>. This target name must be unique across all servers. The target name can be viewed in the WebLogic console or the **config.xml** file.

For each web instance identify:

- Its unique HTTP URL, as seen by the application server. Typically, this uses the same domain as the application server. E.g. http://myApp01.domain.com:7778 where **myWeb01.domain.com** is the DNS entry for the web instance and **7778** is the OHS port responding to HTTP requests (the OTM_PORT on installation).

Scalability can support multiple servers installed on the same physical machine by varying the T3 and HTTP ports setup during installation. It's generally easier, though, to setup a unique virtual DNS entry for each server, regardless of the physical server hosting the application.

Cluster Identification

The standard scalability configuration consists of a single cluster servicing all application servers. The installer stages a cluster named **DEFAULT** for use as a placeholder in non-scalable environments. This cluster should be used to hold the default cluster for scalability.

Server Installation

For each server in the cluster, install an instance of Oracle Transportation Management, setting the host name to the unique host name determined above.

---

[7] Cloud customers are automatically configured with a scalability configuration as pods are added to the system.

[8] Do not use **DEFAULT** as an application server display name. This is reserved as a placeholder in non-scalable implementations.

Property Modifications

> **Note**: The installer creates a glog.properties file with default values for some of these property entries. These default values should modified as explained below.
>
> **Note:** Many of these property entries vary for each web instance and application server. This, for the most part, precludes the use of Property Sets to assign these properties. Changes must be made directly to the glog.properties file.

On each application server, edit the glog.properties file and add the following entries in the **Custom Properties** section:

1. Enable scalability.

```
glog.scalability.on=true
```

2. Register the current server.

```
glog.scalability.thisTarget=[target name]
glog.scalability.thisMachine=[display name]
glog.scalability.thisMachineURL=[t3 URL]
glog.scalability.defaultMachine=[display name]
glog.scalability.defaultMachineURL=[t3 URL]
```

3. Register the default cluster.

```
glog.scalability.defaultServer=DEFAULT
```

4. Register all application servers.

```
!remove glog.scalability.topologyMachineURL
glog.scalability.topologyMachineURL=[T3 URL of application server #1]
glog.scalability.topologyMachineURL=[T3 URL of application server #2]
…
glog.scalability.topologyMachineURL=[T3 URL of application server #n]
```

5. Register all web instances.

```
!remove glog.scalability.web.topologyMachineURL
glog.scalability.web.topologyMachineURL=[HTTP URL of web instance #1]
glog.scalability.web.topologyMachineURL=[HTTP URL of web instance #2]
…
glog.scalability.web.topologyMachineURL=[HTTP URL of web instance #n]
```

6. Suppress the staged server for non-scalability installations.[9]

```
glog.scalability.omitMachines=DEFAULT
```

7. If using Oracle Transportation Management proxy services to communicate with external systems through proxy servers, add the hosts of all application servers to the non-proxy list.

```
glog.integration.http.client.nonProxyHosts=[hostname for application server
#1]|
  [hostname for application server #2]|…|
```

[9] An alternative to the step is to delete the DEFAULT application server via the Application Server manager or via direct SQL statements. Most customers, however, only enable scalability after reaching capacity in a non-scalability implementation. In this situation there are likely foreign key references to the DEFAULT application server. Suppressing the server in the properties files is a more dependable approach to removing it from the scalability network.


```
[hostname for application server #n]|[remainder of the non-proxy hosts]
```

Data Modifications

The following data modifications can be made from within the Oracle Transportation Management user interface[10] or directly to the database via SQL.

8. Add an application server record for each application server registered in the glog.properties file. This requires specifying the following fields:
    a. **Server ID**. The unique display name for the server.
    b. **Server URL**. The T3 URL of the application server. **Note**: this URL must exactly match the URL specified in the properties file.
    c. **Domain Name**. All application servers must be in the **PUBLIC** domain.

    If adding data via the user interface, navigate to **Configuration and Administration > Cluster Management > Application Servers** and add the new application server.
    If adding data via direct SQL, login as **glogdba** and execute the following statements:

    ```
    exec vpd.set_user('DBA.ADMIN');
    insert into app_machine
      (app_machine_gid, app_machine_xid, machine_url, domain_name) values
      ('[Server ID]','[Server ID]','[Server URL]','PUBLIC');
    ```

9. Add each application server to the DEFAULT cluster.

    If adding data via the user interface, navigate to **Configuration and Administration > Cluster Management > Cluster Manager**, edit the DEFAULT cluster and add each server from step #1 to the list of Application Servers servicing the cluster (i.e. the first grid on the manager).

    If adding data via direct SQL, login as **glogdba** and execute the following statements for each application server from step #1:

    ```
    exec vpd.set_user('DBA.ADMIN');
    insert into app_server_machine (app_server_gid, app_machine_gid,
    domain_name)
      values ('DEFAULT','[Server ID]','PUBLIC');
    ```

10. Remove the DEFAULT placeholder application server from the DEFAULT cluster.

    If removing data via the user interface, navigate to **Configuration and Administration | Cluster Management | Cluster Manager**, edit the DEFAULT cluster and delete the DEFAULT Application Server from the list of servers in the cluster.

    If removing data via direct SQL, login as **glogdba** and execute the following statements:

    ```
    exec vpd.set_user('DBA.ADMIN');
    delete from app_server_machine
      where app_server_gid='DEFAULT' and app_machine_gid='DEFAULT';
    ```

Database Modifications

A scalability environment places heavier demands on the database. Each application server requires its own set of connections to process work. With *n* application servers, the maximum connections

[10] While logged in as DBA.ADMIN or a similar user with rights to add PUBLIC data.

allocated by the database, and the number of processes, must be increased *n*-fold. A DBA should resize the Oracle Transportation Management database accordingly before moving to a scalability configuration.

Firewall Modifications

All application servers in the scalability network must be able to access every other application server via its T3 URL. Update any firewalls between servers to allow the T3 port access.

All application servers in the scalability network must be able to access every web instance via its HTTP URL. Update any firewalls between application and web instances to allow the HTTP port access by application servers. All machines outside of the scalability network can be restricted to an HTTP port, but Scalability requires HTTP connections.

When determining if an application server is ready for work, scalability uses two additional ports to determine when the server has completed Oracle Transportation Management startup functions and when it is ready for JMS messages. These two ports are offsets from the T3 ports, where the startup-ready port is based on the **glog.scalability.activatePortOffset** property and the JMS-ready port is based on the **glog.scalability.invokablePortOffset** property. Thus, in addition to allowing every other application server access to the T3 port, the firewall must also allow access to:

- (T3 port) + (glog.scalability.activatePortOffset)
- (T3 port) + (glog.scalability.invokablePortOffset)

These offsets default to **100** and **200**, respectively. Assuming the T3 port were 8001, the firewall should allow access to 8001, 8101 and 8201.

Single Sign-on Modifications

If using Oracle Access Manager or Single Sign-on capabilities, an exception rule or public resource is required for the following URL pattern:

```
glog.webserver.util.SignedServletRouter
```

This is a secure servlet (using basic authentication with a reserved WebLogic user) that provides a single entry point for the application servers to retrieve information from the web instances.

WebLogic Data Source Modifications

All application servers registered with scalability must use the same MDS schema for Oracle Application Developer Framework applications. This requires modifying the WebLogic data source **mds-for-adf**, setup by the installer.

For each application server:

1. Log into the Web Instance WebLogic Domain
2. Click "Services" in the left pane.
3. Click "Data Source" in the right pane.
4. Click "Lock & Edit"
5. Click "mds-for-adf" in the right pane.
6. Click "Connection Pool" tab in the right pane.
7. Change the "URL" to match the URL common MDS database.
8. Change the "Properties" to have the "user=<username>" to point to the common MDS user account.

9. Enter the password for the common MDS user account in the "Password" and "Confirm Password" fields.
10. Click "Save" button.
11. Click "Activate Changes" button.
12. Restart OTM.

Load Balancer Modifications

To ensure the load balancer redirects user requests to healthy servers when an application server is down or unresponsive, the load balancer must support Layer 7 HTTP balancing with health checks. The following servlets are available to properly test web-tier and application-tier health:

- **glog.webserver.test.TestServlet**. This lightweight servlet tests the web instance and its connection to the UI Component of the application server. If either the web instance is down, the application server is down or the application server is unresponsive, the servlet returns an HTTP error.
- **glog.webserver.test.TestTierServlet**. This servlet tests all three tiers of Oracle Transportation Management. It issues a **select 'test' from dual** to the database-tier from the Business Component of the application-tier. If the web instance, application server or database cannot successfully process the request, the servlet returns an HTTP error.

The HTTP load balancer used with Oracle Transportation Management[11] must be configured to issue periodic health checks using one of these servlets. With the new architecture in release 6.4.3, **glog.webserver.test.TestServlet** is the recommended check, as it minimizes the overhead of the check while still testing application server responsiveness.[12] Ideally, both active and passive health checks should be enabled in the balancer, though periodic active checks will suffice.

Restart

After making all the changes above, restart each web and application server registered with scalability. Then follow verification steps listed in Section 6 below.

[11] A load balancer is not included with the system for On Premise customers.

[12] In earlier releases, **TestServlet** would have confirmed only the web server health, not the application server health.

6. Verification and Diagnostics

Once a set of application servers is configured for Scalability, the network needs to be verified. Although the Oracle Transportation Management login page and the other pages may work, it does not necessarily mean that Scalability is working correctly.

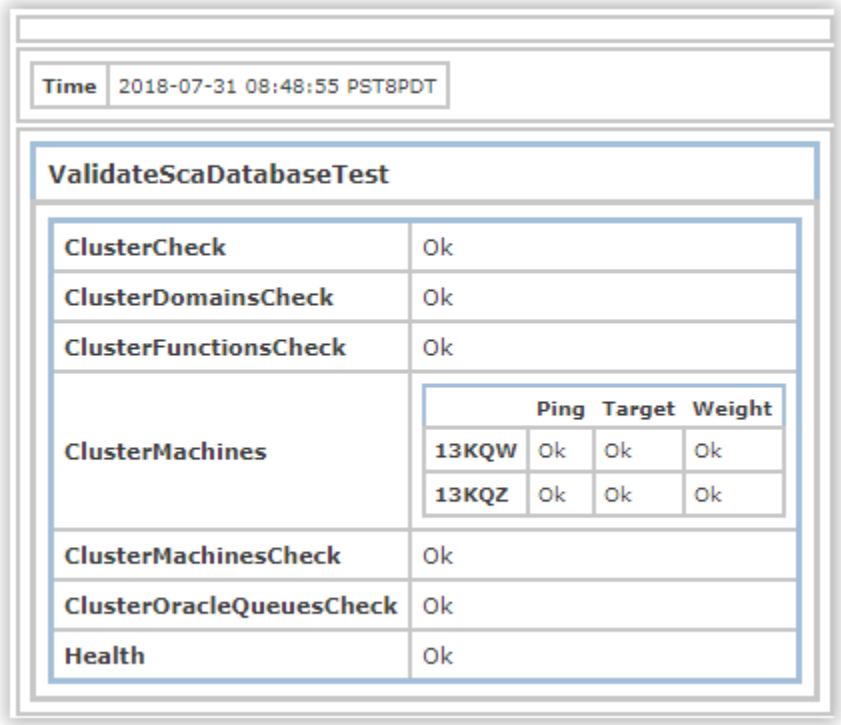
Note that this information is provided primarily for On Premise customers. Scalability for cloud customers is configured and verified automatically during pod installation. Troubleshooting a Cloud Scalability network should be done in tandem with Customer Support.

Summary Verification

A set of Configuration Collectors can be accessed as **DBA.ADMIN** to perform a quick verification of a Scalability setup. Navigate to **Configuration and Administration -> Technical Support -> Configuration Collection** to access **SCALABILITY** collectors.

Verify Scalability Data

First verify the data stored for Scalability is correct. Run the **SCALABILITY DATA** collector. A successful configuration returns:



Time	2018-07-31 08:48:55 PST8PDT

ValidateScaDatabaseTest

ClusterCheck	Ok			
ClusterDomainsCheck	Ok			
ClusterFunctionsCheck	Ok			
ClusterMachines		Ping	Target	Weight
	13KQW	Ok	Ok	Ok
	13KQZ	Ok	Ok	Ok
ClusterMachinesCheck	Ok			
ClusterOracleQueuesCheck	Ok			
Health	Ok			

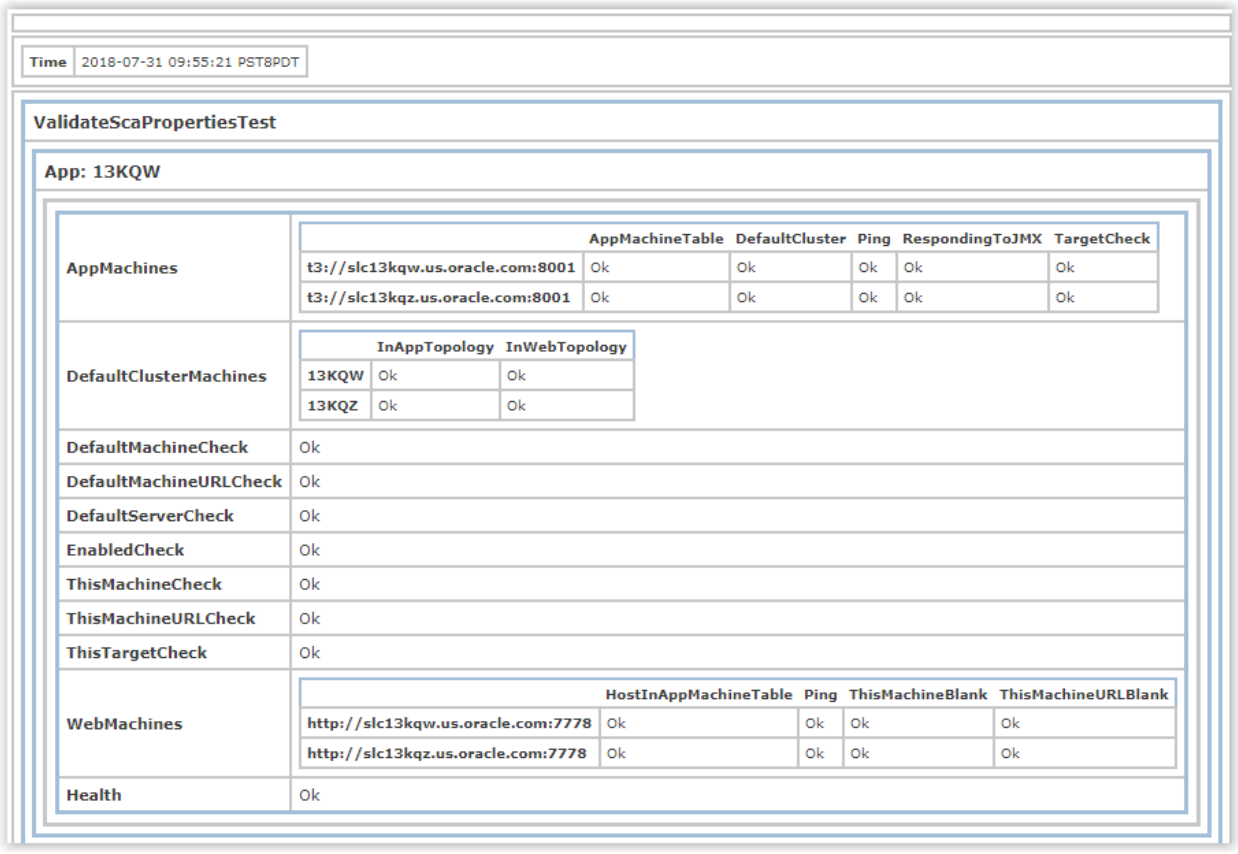
where:

- **ClusterCheck** verifies there is exactly one default cluster
- **ClusterDomainsCheck** verifies domain-based clusters are supported if a domain-based cluster has been configured. (Domain-based clusters are not supported in cloud environments.)
- **ClusterFunctionsCheck** verifies functional clusters are supported if a functional cluster has been configured. (Functional clusters are not supported in cloud environments.)

- **ClusterMachinesCheck** verifies at least one server is assigned to the default cluster.
- **ClusterOracleQueueCheck** verifies Oracle Queue clusters are supported if an Oracle Queue cluster has been configured. (Oracle Queue clusters are not supported in cloud environments.)
- **ClusterMachines** checks that each application server defined in the default cluster is:
    - Responding
    - Configured with a correct, unique Weblogic target name
    - Has a weight of 1.0. Weighted servers have been deprecated in 6.4.3.
- **Health** provides an overall summary of the data check. If health is **OK**, all data validation has been successful.

Verify Scalability Properties

Next, verify properties set for Scalability are correct. Run the **SCALABILITY PROPERTIES** collector. For each application server, a successful configuration returns:



Time	2018-07-31 09:55:21 PST8PDT

ValidateScaPropertiesTest

App: 13KQW

AppMachines		**AppMachineTable**	**DefaultCluster**	**Ping**	**RespondingToJMX**	**TargetCheck**
	t3://slc13kqw.us.oracle.com:8001	Ok	Ok	Ok	Ok	Ok
	t3://slc13kqz.us.oracle.com:8001	Ok	Ok	Ok	Ok	Ok
DefaultClusterMachines		**InAppTopology**	**InWebTopology**			
	13KQW	Ok	Ok			
	13KQZ	Ok	Ok			
DefaultMachineCheck	Ok					
DefaultMachineURLCheck	Ok					
DefaultServerCheck	Ok					
EnabledCheck	Ok					
ThisMachineCheck	Ok					
ThisMachineURLCheck	Ok					
ThisTargetCheck	Ok					
WebMachines		**HostInAppMachineTable**	**Ping**	**ThisMachineBlank**	**ThisMachineURLBlank**	
	http://slc13kqw.us.oracle.com:7778	Ok	Ok	Ok	Ok	
	http://slc13kqz.us.oracle.com:7778	Ok	Ok	Ok	Ok	
Health	Ok					

where:

- **AppMachines** checks each application server defined in the current server's **glog.properties** file.
    - **AppMachineTable** verifies the server is defined in the **APP_MACHINE** table

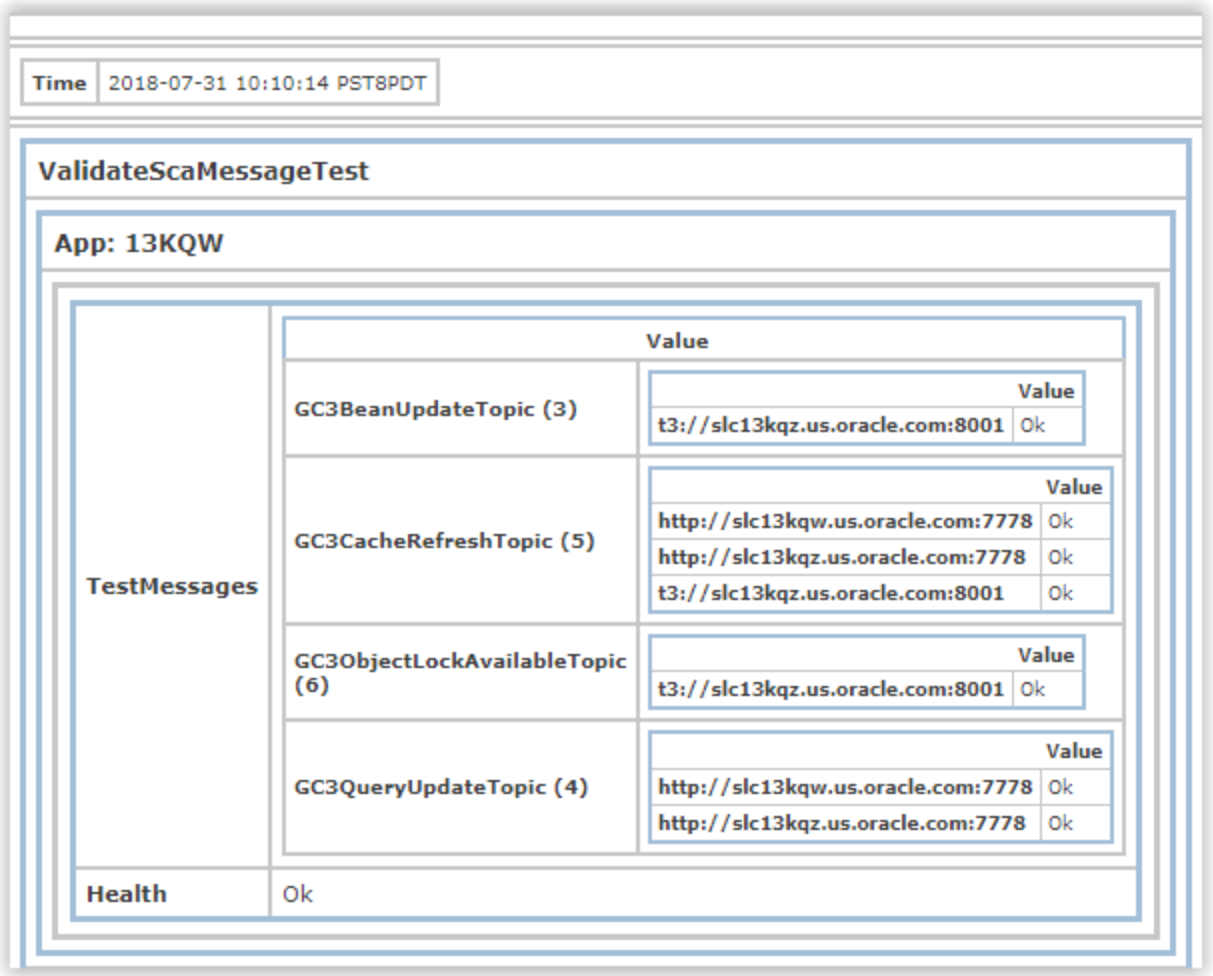
    - **DefaultCluster** verifies the server belongs to the default cluster[13]
    - **Ping** verifies the server is responding
    - **RespondingToJMX** verifies the server can respond to JMX requests
    - **TargetCheck** verifies the server's Weblogic target matches the target defined it its property file.
- **DefaultClusterMachines** verifies each server is listed in the property lists of both application and web instances URLs.
- **DefaultMachineCheck** verifies that each **thisMachine** property matches the **defaultMachine** property. As of 6.4.3, there is no reason to have these properties differ.
- **DefaultMachineURLCheck** verifies that each **thisMachineURL** property matches the **defaultMachineURL** property. As of 6.4.3, there is no reason to have these properties differ.
- **DefaultServerCheck** verifies the **defaultServer** property points to a valid **APP_SERVER** record and has at least one application server (i.e. **APP_MACHINE**) assigned to it.
- **EnabledCheck** verifies that scalability is enabled.
- **ThisMachineCheck** verifies the **thisMachine** property points to a valid **APP_MACHINE** record and the **thisMachineURL** property exactly matches the URL stored on that record.
- **ThisMachineURLCheck** verifies the **thisMachineURL** property matches a valid **APP_MACHINE record** and the **thisMachine** property matches the primary key on that record.
- **ThisTargetCheck** verifies the **thisTarget** property matches the Weblogic target name for the server.
- **WebMachines** checks each web instance[14] defined in the current server's **glog.properties** file.
    - **HostInAppMachineTable** verifies the web-tier host name matches a machine in the **APP_MACHINE** table
    - **Ping** verifies the web instance is responding (e.g. on port 7778)
    - **ThisMachineBlank** verifies that **thisMachine** is set to blank for web-tier properties (i.e. **glog.scalability.thisMachine={web}**
    - **ThisMachineURLBlank** verifies that **thisMachineURL** is set to blank for web-tier properties (i.e. **glog.scalability.thisMachineURL={web}**
- **Health** provides an overall summary of the data check. If health is **Ok**, all property validation has been successful.

Verify Scalability Messaging

Finally, verify JMS messages sent between servers are working correctly for Scalability. Run the **SCALABILITY MESSAGES** collector. For each application server, a successful configuration returns:

[13] This may fail in complex on-premise environments where a server is dedicated only to a particular functional cluster.

[14] Note this will match the physical application server but is checking the web-tier properties setup of the server.



Time	2018-07-31 10:10:14 PST8PDT

ValidateScaMessageTest

App: 13KQW

TestMessages	Value	
	GC3BeanUpdateTopic (3)	Value: t3://slc13kqz.us.oracle.com:8001 Ok
	GC3CacheRefreshTopic (5)	Value: http://slc13kqw.us.oracle.com:7778 Ok; http://slc13kqz.us.oracle.com:7778 Ok; t3://slc13kqz.us.oracle.com:8001 Ok
	GC3ObjectLockAvailableTopic (6)	Value: t3://slc13kqz.us.oracle.com:8001 Ok
	GC3QueryUpdateTopic (4)	Value: http://slc13kqw.us.oracle.com:7778 Ok; http://slc13kqz.us.oracle.com:7778 Ok
Health	Ok	

where:

- **TestMessages** checks the communication of each JMS message type between Scalability components. Specifically:
    - **GC3BeanUpdateTopic** must be successfully sent with a returned acknowledgement to every other application server.
    - **GC3CacheRefreshTopic** must be successfully sent with returned acknowledgement to every other application server and every web instance.
    - **GC3ObjectLockAvailableTopic** must be successfully sent with a returned acknowledgement to every other application server.
    - **GC3QueryUpdateTopic** must be successfully sent with a returned acknowledgement to every web instance.
    - **Health** provides an overall summary of the data check. If health is **Ok**, all property validation has been successful.

Detailed Verification

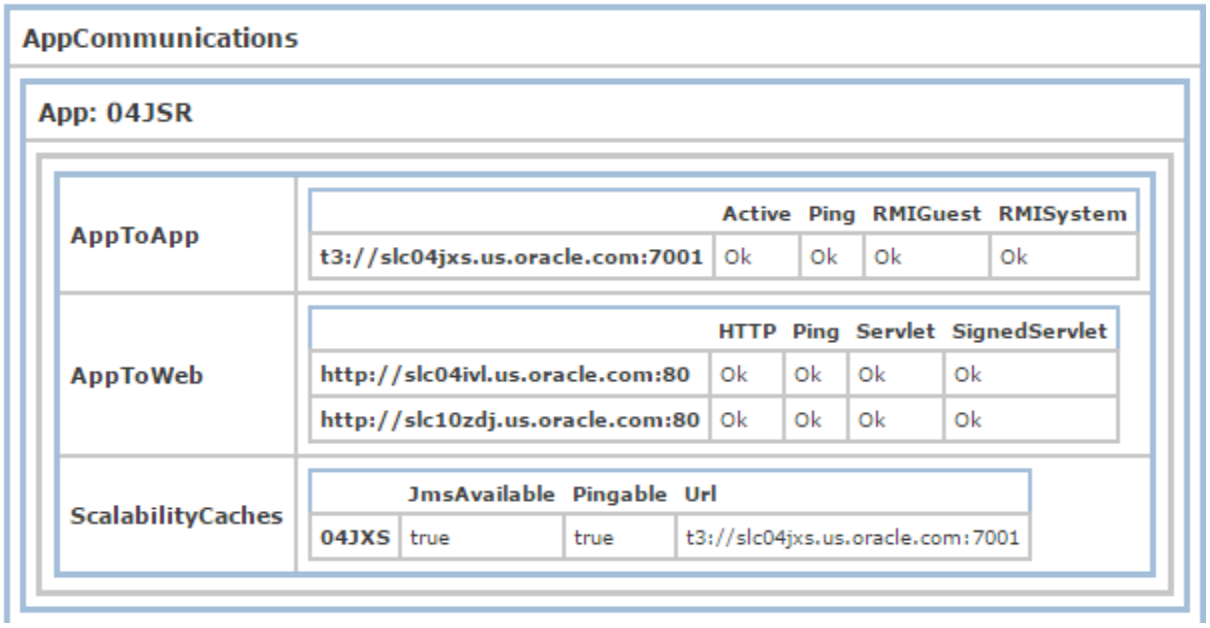
Verify Communication Links

For Scalability to function correctly, each server must be able to communicate with every other server over a number of channels:

- The UI Component on the application server communicates with other application servers via RMI, using JAAS authentication for the **system** or **guest** user.
- The UI Component on the application server communicates with web instances via HTTP, specifically using a signed, authorized servlet.
- The Business Component on the application server communicates with other application servers via RMI, using JAAS authentication for the **system** or **guest** user.
- The Business Component on the application server communicates with web instances via HTTP, specifically using a signed, authorized servlet.

If any of these communication links is configured incorrectly, Scalability performance and diagnostics may be impacted.

To verify proper communication on all of these channels, login as **DBA.ADMIN** (or an equivalent user with access to the DBA menu) and go to **Configuration and Administration** > **Technical Support** > **Performance Collection** from the main menu. Then, collect **APP-TIER-COMMUNICATION** and **WEB-TIER COMMUNICATION** results. Each application server displays a table like:



AppCommunications

App: 04JSR

AppToApp

	Active	Ping	RMIGuest	RMISystem
t3://slc04jxs.us.oracle.com:7001	Ok	Ok	Ok	Ok

AppToWeb

	HTTP	Ping	Servlet	SignedServlet
http://slc04ivl.us.oracle.com:80	Ok	Ok	Ok	Ok
http://slc10zdj.us.oracle.com:80	Ok	Ok	Ok	Ok

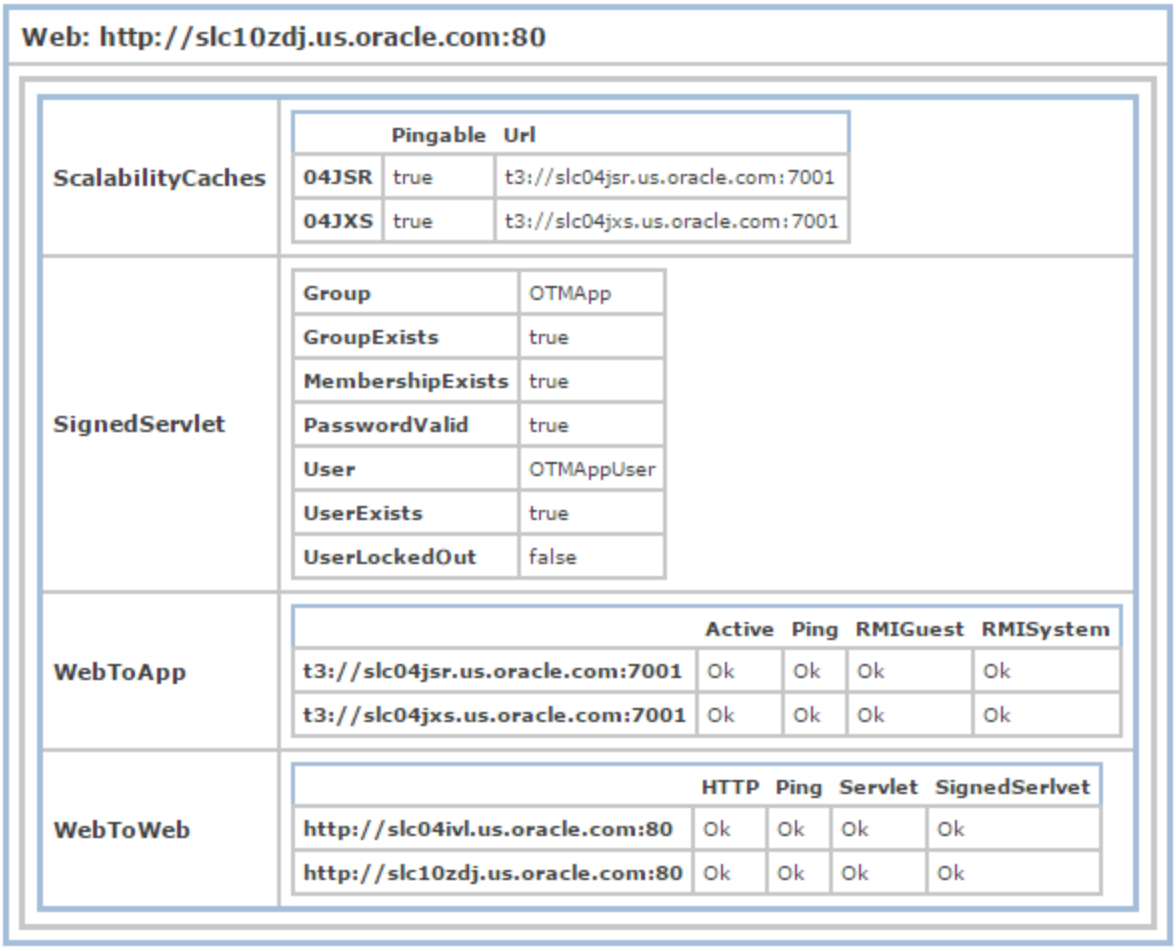
ScalabilityCaches

	JmsAvailable	Pingable	Url
04JXS	true	true	t3://slc04jxs.us.oracle.com:7001

Figure 6-1: APP-TIER COMMUNICATIONS Diagnostic

where the **AppToApp** rows show the communication status to each other application server over RMI; **AppToWeb** rows show the communication status to each web instance over HTTP. All columns should show **Ok** if the servers are up. Any errors imply the application server could not successfully communicate with a specific application server or web instance.

Each web instance displays a table like:



Web: http://slc10zdj.us.oracle.com:80

ScalabilityCaches

	Pingable	Url
04JSR	true	t3://slc04jsr.us.oracle.com:7001
04JXS	true	t3://slc04jxs.us.oracle.com:7001

SignedServlet

Group	OTMApp
GroupExists	true
MembershipExists	true
PasswordValid	true
User	OTMAppUser
UserExists	true
UserLockedOut	false

WebToApp

	Active	Ping	RMIGuest	RMISystem
t3://slc04jsr.us.oracle.com:7001	Ok	Ok	Ok	Ok
t3://slc04jxs.us.oracle.com:7001	Ok	Ok	Ok	Ok

WebToWeb

	HTTP	Ping	Servlet	SignedSerlvet
http://slc04ivl.us.oracle.com:80	Ok	Ok	Ok	Ok
http://slc10zdj.us.oracle.com:80	Ok	Ok	Ok	Ok

Figure 6-2: WEB-TIER COMMUNICATIONS Diagnostic

where the **WebToApp** rows show the communication status to each application server over RMI; **WebToWeb** rows show the communication status to each web instance over HTTP. All columns should show **Ok** if the servers are up. Any errors imply the web instance could not successfully communicate with a specific application server or web instance.

Verify JMS Connections

To verify JMS communications are correctly registered between application servers and components, perform the following functions once all servers are running:

- Edit an Involved Party Qualifier
- Activate and Deactivate a Workflow Agent

Then, as DBA.ADMIN[15], navigate to the JMS Message Diagnostic: **Configuration and Administration > Technical Support > Diagnostics and Tools > Scalability > JMS Messages**. Select **Show Servers** and **Show Web Servers** on the diagnostic screen.

[15] Or a user with the same security access as DBA.ADMIN

The diagnostic will show JMS messages sent from one server and received by others. A correctly configured network will show:

- At least one **ActivateServer** message sent from an application server to every other application server.
- At least one **QueryUpdate** message sent from an application server to each web instance.
- At least two **CacheRefresh** messages sent from an application server to each web instance and other application server.

As messages are only received when a server is running, and only sent if at least two servers are running, messages may be missed prior to server startup. The number of sent messages from one server may not precisely equal the number of received messages on another. This diagnostic validates JMS communication links as long as each application server shows sent and received messages; each web instance shows received messages.

Note the Message Diagnostic Servlet can also be used to monitor JMS performance providing statistics on JMS latency and processing.

Message Diagnostic Servlet

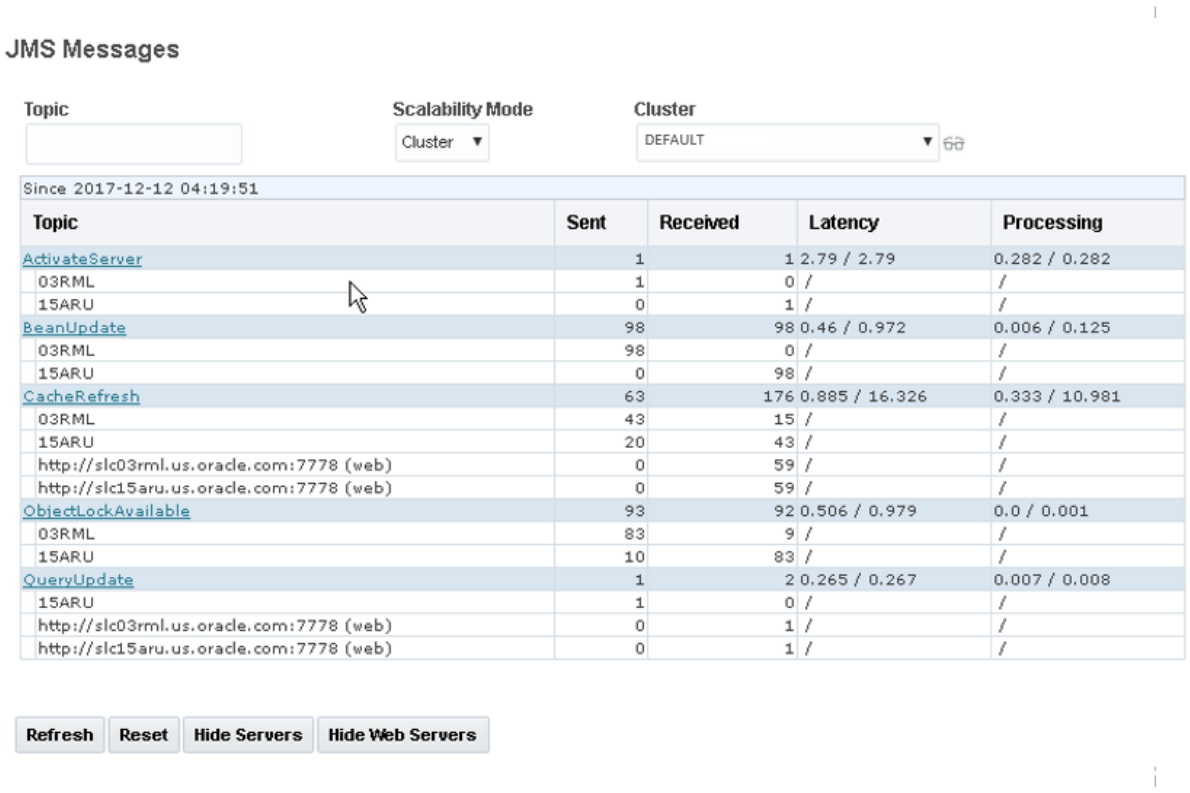


Figure 6-3: Message Diagnostic Servlet

Resolving Issues

If any of the verification steps above fails, review the Configuration section above. Common errors include:

- Missing properties or typographical errors in properties. Check all **glog.scalability** properties for accuracies – on each application server.

- Unique host names were not specified for one or more application servers. This leads to overlapping WebLogic target names. Either reinstall the application servers with unique host names or modify the WebLogic **config.xml** file to replace all instances of **gc3webapp-<host>** with a unique host name. Update the properties file for the modified servers to point to the updated WebLogic target.
- The URL added for an Application Server in the user interface (i.e. the MACHINE_URL in the APP_MACHINE table) does not exactly match the URL specified in the properties file. Modify the URL in the user interface / database or adjust property files to match.
- Application Server ports are blocked by a firewall. This can include the T3 port, the activation port or the invokable port. Adjust the firewall to allow access to these ports from all application server machines.
- Web Server ports are blocked by a firewall. The HTTP port must be accessible by all web and applications servers. Adjust the firewall to allow access to these ports from all application server machines.
- HTTP access has been disabled in Oracle HTTP Server. Configure OHS to support HTTP, optionally restricting access to the application and web instances.
- A single-sign on system, firewall, load balancer or custom servlet filter is redirecting or restricting access to the Signed Servlet router (i.e. **glog.webserver.util.SignedServletRouter**). Add an exception to provide access to this secure servlet and avoid any SSO challenges.
- The password specified on installation for Weblogic admin user has expired. Refer to the security guide for instructions to reset the admin user password.
- Application Functions or Domains have been assigned to the DEFAULT cluster. The DEFAULT cluster must not be dedicated to any explicit functionality or domains.
- A non-DEFAULT cluster has no Application Functions or Domains assigned to it. This effectively creates two default clusters and corrupts the Scalability configuration.
- The **glog.scalability.activatePortOffset** or **glog.scalability.invokablePortOffset** properties are set to 0. This breaks the Scalability requirement for distinct activation and invokability ports on the application server.

7. Additional Recommendations

Integration Transmission Redo Process

In any Oracle Transportation Management environment, there should be an integration transmission redo process implemented unless the upstream external application will resend integrations.

By setting up this the Redo Transmission Processing, an Oracle Transportation Management client will be able to reprocess integrations that have been stored but are not processed yet. This process is required because these stored integrations will not be processed on a restart.

In order to schedule Redo Transmission Processing:

13. Go to Business Process Automation > Process Management > Redo Transmission Processing.
14. Select a Transmission Type of Inbound.
15. Set N as the Redo Number of Inbound Transmissions.
16. Schedule the process to run every T minutes. Typically, you can set N=50, T=5 minutes.

Thus, once you run the update command above, N uncompleted transmissions will be reprocessed on Application Server A within T minutes. N more will be reprocessed at each T minute interval. As long as T is not too frequent, the overhead of scanning for REDO transmissions is minimal: the status column is indexed and histogrammed so querying the status is optimized.

Note that Redo processing relies on polling the transmission table for transmissions needing reprocessing, already assigned to the current Application Server.

Object Lock Cleanup Process

In a Scalability environment, Oracle Transportation Management can accumulate millions of records in the OBJECT_LOCK table. A workflow process, **Object Lock Cleanup**, is available to remove these old and un-owned object lock records. By default, it runs every day at 0500 UMT, cleaning up object locks not used in 30 days. This can and should be modified to fit individual client timetables and business processes. If you do not currently have this default recurring process on your particular Oracle Transportation Management version, it is strongly advised to create your own recurring process to accomplish the object lock record cleanup process. Note that periodically cleaning up unused object locks is critical to performance in a Scalability implementation.

> **Note:** It is strongly recommended the client does not adjust the day interval to a very high number for the recurring process.

Server Removal

When an application server in the Scalability topology will no longer be part of the topology because of hardware failure or extended time down, it's important to re-route failed integration from the server to another application server.

A Process Control topic, **Configuration and Administration > Process Management > Report Server Failure**, releases resources held by a specified application server, including:

- **Transmissions**. Fresh transmissions owned by the application server are set to Redo. The Redo Transmission Processing process running on other machines in the cluster will pick up these transmissions.

- **OBJECT_LOCK locks held by the application server are released**. This also includes cleaning up any mediator/object locks. Other application servers are then notified that object locks are now available.

This allows for transmissions running on the failed machine to be processed by the active application server(s) in the cluster. This will also allow for transmissions blocked on the active machines to run to completion.

> **Note:** The topic XML for the "Report Server Failure" can also be sent in via integration. This would allow for an external monitoring system to send this XML automatically if it detected a JVM crash, an application server crash, or hardware failure.

Web Services

Scalability does not handle the load balancing of requests using web services; the application communicating with the Oracle Transportation Management application servers will have to load balance the web service requests itself.

In addition to not load balancing web service requests, the application server port may be available (pingable) before the Oracle Transportation Management application is ready. This means that the external application may start sending web service requests before Oracle Transportation Management is fully activated. To solve this, the Oracle Transportation Management application opens a new port when it is fully activated and ready to receive web service requests. It is recommended to ping this new port from the external application before sending requests. This port is essentially an offset port from the application server port. The default port is 8101 for WebLogic. This port can be controlled via a property. Please see Advanced Scalability Properties for more details on this setting.

8. Reference

Scalability Properties

These Scalability properties need to be set correctly for a proper Scalability implementation.

Table 8-1: Scalability Properties

Scalability Properties
`glog.scalability.on` The glog.scalability.on property turns scalability on or off. **Usage**: `glog.scalability.on=[true | false]`
`glog.log.ID.JMS.on` The glog.log.ID.JMS.on property turns the JMS logging on or off. **Usage**: `glog.log.ID.JMS.on=[true | false]`
`glog.log.ID.Scalability.on` The glog.log.ID.Scalability.on property turns the Scalability logging on or off. **Usage**: `glog.log.ID.Scalability.on=[true | false]`
`glog.scalability.thisTarget` The glog.scalability.thisTarget property is the target name of the WebLogic Application Server that this property will be used on. This name of the application server must be unique. **Usage**: `glog.scalability.thisTarget=[gc3webapp-APP01]`
`glog.scalability.thisMachine` The glog.scalability.thisMachine property is the unique display name of this application server. This name must match the data in the APP_MACHINE_GID column of database table, APP_MACHINE. The following values are only examples. **Usage**: `glog.scalability.thisMachine=[ APP-01 | APP-02 | … | APP-0n ]`
`glog.scalability.thisMachineURL` The glog.scalability.thisMachineURL property is the URL of this current application server. This URL needs to match exactly the URL data that is in the MACHINE_URL column of the database table of APP_MACHINE. **WebLogic Usage**: `glog.scalability.thisMachineURL=t3://app01.domain.com:8001`

Scalability Properties

`glog.scalability.defaultServer`

The glog.scalability.defaultServer property is the ID of the default cluster. This ID must match the data in the APP_SERVER_GID column of the database table, APP_SERVER.

Usage: `glog.scalability.defaultServer =[DEFAULT | … ]`

`glog.scalability.defaultMachine`

The glog.scalability.defaultMachine property is the ID of the default Machine. For an application instance, this ID is simply the application machine ID for the current server. For a web instance instance, the ID should be set to one of the application servers in the default cluster. It must match the data in the APP_MACHINE_GID column of the database table, APP_MACHINE.

Usage: `glog.scalability.defaultMachine =[APP01 | … ]`

`glog.scalability.defaultMachineURL`

The glog.scalability.defaultMachineURL property is the URL for an application server in the default cluster. This URL needs to match exactly the URL data that is in the MACHINE_URL column of the database table of APP_MACHINE.

WebLogic Usage: glog`.scalability.defaultMachineURL =t3://app01.domain.com:8001`

`glog.scalability.topologyMachineURL`

The following is the list of properties for all of the available application server(s). These are only used by the web instance(s) to poll for network topology. Remember to put only one of these properties per line.

WebLogic Usage:

```
!remove glog.scalability.topologyMachineURL
glog.scalability.topologyMachineURL=t3://app01.domain.com:8001
glog.scalability.topologyMachineURL=t3://app02.domain.com:8001
```

`glog.scalability.web.topologyMachineURL`

The following is the list of properties for all of the available web instance(s). These are only used by the web and application server(s) to retrieve web-tier diagnostics and notify web machines of topology changes. Remember to put only one of these properties per line.

WebLogic Usage:

```
!remove glog.scalability.web.topologyMachineURL
glog.scalability.web.topologyMachineURL=http://app01.domain.com:7778
glog.scalability.web.topologyMachineURL=http://app02.domain.com:7778
```

The following properties are used to control the collection associated JMS data, which is displayed in the Message Diagnostic Servlet.

Table 8-2: Collection Associated with JMS Data

JMS Data Collection Properties

`glog.scalability.topic.trackUpdates`
`glog.scalability.topic.trackUpdates=[ true | false ]`
 	
If true, track general message statistics

`glog.scalability.topic.trackLatency`
`glog.scalability.topic.trackLatency=[ true | false ]`
 	
If true, track message latency

`glog.scalability.beanUpdate.trackUpdates`
`glog.scalability.beanUpdate.trackUpdates=[ true | false ]`
 	
If true, track statistics specific to BeanUpdateTopic

`glog.scalability.cacheRefresh.trackUpdates`
`glog.scalability.cacheRefresh.trackUpdates=[ true | false ]`
 	
If true, track statistics specific to CacheRefreshTopic

`glog.scalability.queryUpdate.trackUpdates`
`glog.scalability.queryUpdate.trackUpdates=[ true | false ]`
 	
If true, track statistics specific to QueryUpdateTopic

The following properties are used to better diagnose issues occurring in a scalability environment. They are optional.

Table 8-3: Properties to diagnose issues occurring in a scalability environment

Diagnostic Properties
glog.exception.sequencePrefix=<short log prefix> This is used in conjunction with glog.exception.sequence=all to tie error messages seen on the UI to the underlying cause in web-tier or application-tier log files. If exception sequencing is enabled, the sequence prefix allows exceptions tags to be unique across web and application servers. For example, assume there are two web instances: W1 and W2; two application servers: A1 and A2. The following properties would provide distinct error ID's to the user: • On W1: glog.exception.sequencePrefix=w1 • On W2: glog.exception.sequencePrefix=w2 • On A1: glog.exception.sequencePrefix=a1 • On A2: glog.exception.sequencePrefix=a2 For greater legibility in the logs, we recommend keeping the sequence prefixes short.

Advanced Scalability Properties

These are a set of advanced scalability properties. These should be modified with extreme caution, and with a lot of testing before moving into a production environment.

Table 8-4: Advanced Scalability Properties

Advanced Scalability Properties
`glog.scalability.activatePortOffset` The glog.scalability.activatePortOffset property is port offset from the RMI port of the application server that should be used in any Oracle Transportation Management environment for determining if Oracle Transportation Management is fully initialized and activated. This property should be used particularly for web service integration requests. A third party application should ping the (RMI port + this property offset value) port to determine when the server is ready for integrations. This property defaults to 100. **Usage**: `glog.scalability.activatePortOffset=100`
`glog.scalability.invokablePortOffset` A keep-alive ping exists for application servers to account for missed JMS synchronization on startup. The keep alive relies on an invocation port ping on each application server. This port is set up after all other startup classes have activated (including JMS messaging). The port is controlled by this property, which is the number offset from the primary WebLogic port (e.g. 8001). It defaults to 200. At each keep-alive ping, the application server invokes all other pingable application servers to: 1. Make sure the source server is in the scalability map. 2. Make sure JMS connections exist between the source and destination servers.. **Usage**: `glog.scalability.invokablePortOffset=200`

`glog.scalability.keepAliveInterval`
 	
The glog.scalability.keepAliveInterval property controls the number of seconds between the keep alive pings from an application server to the web instances and other application servers. This property is helpful when a server thinks an application server is down due to unresponsiveness. This ping will re-establish the connection between the application server and other servers. The default is 300 seconds (5 minutes).
 	
`Usage: glog.scalability.keepAliveInterval=300`

`glog.scalability.restrictToServers` The glog.scalability.restrictToServers property controls which application clusters will be used in the scalability topology. This property will restrict scalability the clusters it will use. This property is not recommended for a client production environment, but could be useful in a testing environment. **Usage**: `glog.scalability.restrictToServers=DEFAULT`

Advanced Scalability Properties

`glog.scalability.restrictToMachines`

The glog.scalability.restrictToMachines property controls which application servers will be used in the scalability topology. This property will restrict Scalability the application servers it will use. This property is not recommended for a client production environment, but could be useful in a testing environment.

Usage: `glog.scalability.restrictToMachines=DEFAULT`

`glog.scalability.omitServers`

The glog.scalability.omitServers property controls which clusters should be omitted from the scalability topology. This allows clusters defined in the database to be ignored by the implementation. This property is not recommended for a client production environment, but could be useful in a testing environment.

Usage: `glog.scalability.omitServers=BULKPLAN`

`glog.scalability.omitMachines`

The glog.scalability.omitServers property controls which application servers should be omitted from the scalability topology. This allows servers defined in the database to be ignored by the implementation. In general, this property is not recommended for a client production environment, but could be useful in a testing environment. Its use is recommended, however, to suppress the DEFAULT application server staged for a non-Scalability implementation.

Usage: `glog.scalability.omitMachines=DEFAULT`

`glog.objectLock.delayForScalability`

The glog.objectLock.delayForScalability property controls the number of milliseconds a scalability application server should wait before trying to obtaining an object lock. This time delay will give other application servers a greater chance of obtaining the object lock. By default, a scalability application server releasing a lock will naturally favor any lock waiters on the same JVM. This occurs because the object lock release operation notifies the local mutex, updates the database and sends out the JMS notification. It is extremely likely that any local lock waiters will wake up and obtain the lock before any of the other Scalability application servers get a chance.

Usage: `glog.scalability.delayForScalability=2000`

`glog.scalability.debugWebApp`

This property provides very low level debug information in a Scalability environment. This would need to be set in all of the glog.properties files on all of the webserver(s) and application server(s) in the topology. This is not recommended to be set in a production environment without first being directed by support to set this.

Usage: `glog.scalability.debugWebApp=true`

This property may be set using the `CUSTOM` property set.

