ORACLE DATABASE 12c

# Design and Deploy Tomcat Applications for Planned, Unplanned Database Downtimes and Runtime Load Balancing with UCP

## In Oracle Database RAC and Active Data Guard environments

ORACLE WHITE PAPER NOVEMBER 2017

Table of Contents

## Introduction

Achieving maximum application uptime without interruptions is a critical business requirement. There are a number of requirements such as outage detection, transparent planned maintenance, and work load balancing that influence application availability and performance. The purpose of this paper is to help Java Web applications deployed with Apache Tomcat, achieve maximum availability and scalability when using Oracle.

Are you looking for best practices to hide your tomcat applications from database outages? Are you looking at, smooth & stress-free maintenances of your tomcat applications? Are you looking at leveraging Oracle Database's runtime load balancing in your tomcat web applications? This paper covers the configuration of your database and tomcat servlets for resiliency to planned maintenance, unplanned database outages, and dynamic balancing of the workload across database instances, using RAC, ADG, GDS[1], and UCP.

## Issues to be addressed

The key issues that impede continuous application availability and performance are:

- **Planned Maintenance:**
  - **Achieve transparent maintenance:** Make the maintenance process fast and transparent to applications for continuous availability.
  - **Session draining:** When the targeted instance is brought down for maintenance, ensure that all work completes. We will describe how to drain sessions without impacting in-flight work and also avoid logon storms on active instance(s) during the planned maintenance.
- **Unplanned Downtimes:**
  - **Outage detection:** Web application's timeouts are unpredictable and unreliable. This paper describes how to configure Tomcat servlets to be notified of outages as fast as possible.
  - **Error handling:** Several types of SQL exceptions may be received by your servlets; how to determine that such errors are indicative of database service failure?
  - **Recovery with Response Time Targets:** Upon outage, the Oracle Database RAC system needs a short period of time to recover before becoming fully operational again. How to react quickly and keep such "*brownout*" period under SLA targets?
  - **Outcome of in-flight work:** Have you ever paid twice for books, flights, or taxes? Making a reliable determination of the outcome of the in-flight transaction in the face of database outages was a challenge until Oracle Database 12c. We will describe, how to design servlets and configure Oracle Database 12c for solving this challenge.
  - **Continuation of in-flight work:** How to design servlets and configure Oracle Database 12c and UCP to allow safe and transparent replay of in-flight transactions in the event of unplanned database outages.
- **Workload Balancing**: In RAC, RAC ONE, and ADG environments, connection requests are by default distributed randomly by the Net Listener. How to configure your web applications and configure the database for optimal distribution of the workload when the node/services are added/removed?

1 http://www.oracle.com/technetwork/database/availability/maa-consolidation-2186395.pdf

If you are a Java developer or architect looking to exploit Oracle Database Real Application Cluster (RAC), Exadata, Active Data Guard (ADG), Global Data Services (GDS) and Oracle Universal Connection Pool (UCP) solutions for planned and unplanned database downtime and workload balancing, this is the paper for you. If you are a DBA looking for database configuration steps to meet Web apps high-availability and load balancing requirements, this is the paper for you too.

## Oracle Database 12c High-Availability and Load Balancing Concepts

To support high-availability and load balancing solutions, Oracle Database 12c Release 2 (12.2) and prior releases furnish HA configurations (RAC, Data Guard) and features which are leveraged by Oracle Database drivers (e.g., Oracle JDBC) and connection pools (e.g., UCP). This paper will refer to the following features, mechanisms, and concepts described in **Java Programming with Oracle Database 12c RAC and Active Data Guard** [2] white paper:

- Universal Connection Pool (UCP)
- Fast Application Notification (FAN)
- Oracle Notification Service (ONS)
- Fast Connection Failover (FCF)
- Logical Transaction ID (LTXID)
- Database Request
- Recoverable Errors
- Mutable Functions
- Transaction Guard (TG)
- Application Continuity (AC)

## Configure Tomcat for UCP

Universal Connection Pool (UCP) has the built in support for planned maintenance, unplanned downtimes, and runtime load balancing. UCP along with RAC, RAC One, and ADG is a tested and certified combination for handling database failovers. UCP has been successfully used by many customers to handle failovers seamlessly. Configuring UCP in Apache Tomcat is explained in detail, hereafter.

Deploying a servlet which accesses Oracle Database through Oracle JDBC driver and Oracle Universal Connection Pool (UCP) in a Tomcat application container requires the following steps:

- Create a New Database Resource
- Create a JNDI lookup in the servlet
- Create a *web.xml* for the Servlet

### Create a New Database Resource

First ensure that Oracle JDBC jar ojdbc8.jar[3], ucp.jar, and ons.jar from the version 12.2 are present in $CATALINA_HOME/lib. Make sure that all three jar files are from the same database version. DO NOT mix the version of these jars. Always, try to use the latest JDBC driver and UCP in order to leverage the latest capabilities and performance

2 http://www-content.oracle.com/technetwork/database/application-development/12c-ha-concepts-2408080.pdf

3 http://www.oracle.com/technetwork/database/features/jdbc/jdbc-ucp-122-3110062.html

improvements. UCP is configured either as a global resource in **server.xml** for all applications, or as an application specific resource in **context.xml** ($CATALINA_HOME/webapps/<APPLICATION_NAME>/META-INF/context.xml) as shown in Fig 1a below.

Note that all required UCP properties such as minPoolSize, maxPoolSize, are mentioned while creating a new database resource. Two connection properties and their settings related to high availability are summarized here.

(a) *fastConnectionFailoverEnabled (a.k.a FCF)*: The12.2 UCP has FCF enabled by default. However, FCF should be set to true (fastConnectionFailoverEnabled=true) when using UCP 12.1.0.2,12.1.0.1 and 11.2.0.4.

(b) *ONSConfiguration:* The 12.2 UCP has auto-ONS capability and receives the ONS notifications automatically. However, if you are using pre 12.1 RAC database, this property must be set explicitly.
`Example: onsConfiguration="nodes=<RAC_node1>:<port1>, <RAC_node2>:<port2>, <RAC_node3>:<port3>`

**Fig 1a: Database Resource in Tomcat with 12.2 UCP**

```
<Context docBase="UCPTomcat" path="/UCPTomcat" reloadable="true"
source="org.eclipse.jst.jee.server:UCPTomcat">
<Resource
   name="tomcat/UCPPool"
   auth="Container"
<!-- Defines UCP or JDBC factory for connections -->
   factory="oracle.ucp.jdbc.PoolDataSourceImpl"
<!-- Defines type of the datasource instance -->
   type="oracle.ucp.jdbc.PoolDataSource"
   description="UCP Pool in Tomcat"
<!-- Defines the Connection Factory to get the physical connections -->
   connectionFactoryClassName="oracle.jdbc.pool.OracleDataSource”
   minPoolSize="2"
   maxPoolSize="60"
   initialPoolSize="15"
   autoCommit="false"
   user="scott"
   password="tiger"
<!-- FCF is auto-enabled in 12.2.  Use this property only if you are using Pre 12.2 UCP
   fastConnectionFailoverEnabled=”true”  -->
<!-- Database URL -->
url="jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=tcp)(HOST=proddbclust
er-scan)(PORT=1521)))(CONNECT_DATA=(SERVICE_NAME=proddb)))"
</Resource>
 </Context>
```

**Create a JNDI lookup in the servlet**

The following code snippet shows how to get a database connection by referring to the JNDI datasource created in Tomcat.

```
Context ctx = new InitialContext();
Context envContext = (Context)ctx.lookup("java:/comp/env");
```

```
// Look up a data source
javax.sql.DataSource ds=(javax.sql.DataSource)envContext.lookup ("tomcat/UCPPool");
PoolDataSource pds= (PoolDataSource)ds;

Connection conn = pds.getConnection();
```

**Create a web.xml for the servlet**

The data source resource reference should also be present in *web.xml* as illustrated hereafter.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" version="3.0">

  <servlet>
    <servlet-name>DemoServlet</servlet-name>
    <servlet-class>DemoServlet</servlet-class>
  </servlet>
  <resource-ref>
    <description>UCP datasource to connect to DB</description>
    <res-ref-name>tomcat/UCPPool</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
  </resource-ref>
</web-app>
```

## Hiding Planned Maintenance from Tomcat Servlets

For maintenance purposes (i.e., software upgrades), the Oracle Database instances can be gracefully shutdown one at a time without disrupting the operations and availability of web applications. Upon FAN **DOWN** event[4], UCP drains sessions away from the instance(s) targeted for maintenance. What is the configuration of web applications and the database to achieve *session draining at service stop or relocation?* In a nutshell, the procedure consists in stopping non-singleton services running on the target database instance or relocating singleton ones from the target instance to a new instance.

**Web Applications Steps**

To hide the planned database maintenance, web applications need to:

(i) Make sure you are using UCP as a connection pool. The JDBC driver *ojdbc8.jar* and *ucp.jar* should be present in the classpath of an application.

(ii) Check that *ons.jar* is also in the classpath of an application.

(iii) Enable Fast Connection Failover (FCF).

The 12.2 UCP will auto-enable Fast Connection Failover (FCF). However, if you are using Pre 12.2 UCP then, FCF needs to be enabled explicitly as a pool property. FCF can be enabled either in *context.xml* (recommended method) or set programmatically. Refer to *Fig 1a. Database Resource in Tomcat with 12.2 UCP* for more details on enabling FCF in *context.xml.*

FCF can also be enabled programmatically as shown below.

```
PoolDataSource pds = new PoolDataSourceFactory.getPoolDataSource();
```

4 status=down reason=user

```
pds.setFastConnectionFailoverEnabled(true);
// not required with auto-ONS in 12c
pds.setONSConfiguration("nodes=<RACNode1>:<port1>,<RACNode2>:<port2>,<RACNode3>:port3");
```

(iv) Enable draining through server side knob or client side knob.

In 12.2, for graceful draining of database sessions, use the server side *drain_timeout* attribute. It can be set using the commands as specified below. Also, set *failover_restore* to *level1*

```
srvctl modify service -db <db_name> -service <svc_name> -drain_timeout <timeout_value>
srvctl modify service -db <db_name> -service <svc_name> -failover_restore level1
```

In pre-12.2 UCP, use a client side setting for handling graceful draining. The property `oracle.ucp.PlannedDrainingPeriod` can be specified as a Java system property (i.e., using –D) or it can be set as a context parameter.

```
-Doracle.ucp.PlannedDrainingPeriod=30
```

Setting it as a context parameter:

```
<!-- System property for graceful planned maintenance -->
<context-param>
<param-name>oracle.ucp.PlannedDrainingPeriod</param-name>
<param-value>30</param-value>
</context-param>
```

**DBA or RDBMS Steps**

DBAs must perform the following steps[5] to stop all services on the target machine where the database instance is scheduled for maintenance. For each service repeat the following actions:

1. Stop the service without using `-force` option or relocate the service. Service relocation is required for singleton service (i.e., runs only on one instance at a time)

```
$srvctl stop service -db <db_name> -service <service_name> -instance <instance_name
or (NOTE: Omitting -service stops all services)
$srvctl relocate service -db <db_name> -service <service_name> -oldinst <oldins> -
newinst <newinst>
```

5 See Metalink note 1593712.1 @ https://support.oracle.com/epmos/faces/DocumentDisplay?id=1593712.1 for more details

2. Disable the service and allow sessions some time to drain. E.g., 2-30 minutes. This avoids the logon storm on the other active instance where the workload gets transferred. Disabling service is optional if you choose to disable the instance.

```
$srvctl disable service –db <db_name> -service <service_name> -instance <instance_name>
```

3. Wait to allow sessions to drain Example: 10-30 minutes
4. Check for long-running sessions and terminate these (you may check again afterwards)

```
SQL> select count(*) from ( select 1 from v$session where service_name in
upper('<service_name>') union all
select 1 from v$transaction where status = 'ACTIVE' )
SQL> exec dbms_service.disconnect_session ('<service_name>',
DBMS_SERVICE.POST_TRANSACTION);
```

5. Repeat steps 1-4 for all services targeted for planned maintenance.
6. Stop the database instance immediately.

```
$srvctl stop instance –db <db_name> -instance <instance_name> -stopoption immediate
```

7. Disable instance to prevent restarts during maintenance

```
$srvctl disable instance –db <db_name> -instance <instance_name>
```

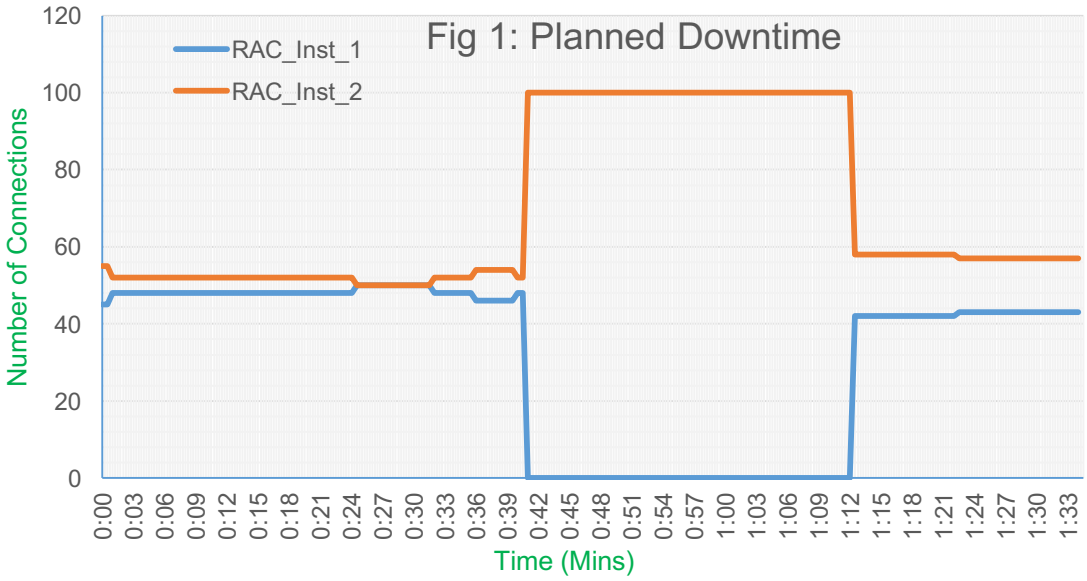
8. Apply patch or carry out the scheduled maintenance work
9. Enable then Start the instance again

```
$srvctl enable instance –db <db_name> -instance <instance_name>
$srvctl start instance –db <db_name> -instance <instance_name>
```

10. Enable then start the service back and check if the service is up and running

```
$srvctl enable service –db <db_name> -service <service_name> -instance <instance_name>
$srvctl start service –db <db_name> -service <service_name> -instance <instance_name>
```

Fig 1, hereafter shows connections distribution of XYZ service across two RAC instances before and after Planned Downtime. Notice that the connection workload went from fifty-fifty across both instances to hundred-zero. In other words, RAC_INST_1 can be taken down for maintenance without any impact on the business operation.




## Hiding Unplanned Database downtime from Tomcat Servlets

Tomcat Servlets can be configured to handle unplanned database outages using the following features and mechanisms:

- **Fast Connection Failover (FCF)**
- **Transaction Guard (TG)**
- **Application Continuity (AC)**

Please refer to the white paper, **Java Programming with Oracle Database 12c RAC and Active Data Guard** [6] for understanding these concepts in detail.

**Developer or Web Application Steps**

To hide the unplanned database outages, web applications need to do the following:

(i) Use the replay data source

Example: In Apache Tomcat, `connectionFactoryClassName` defined while creating a datasource using UCP should use the replay (`oracle.jdbc.replay.OracleDataSourceImpl`) datasource.

(ii) Enable Fast Connection Failover (FCF)

FCF enables UCP to detect dead instance and helps in transferring the work load to the surviving active instance as soon as the unplanned down event occurs. In case of UCP 12.2, FCF is auto-enabled. However, if you are using pre-12.2 UCP, then enable FCF explicitly (Refer to “Hiding Planned Maintenance using Tomcat Servlets).

(iii) Enable Transaction Guard (TG) and Application Continuity (AC)

6 http://www-content.oracle.com/technetwork/database/application-development/12c-ha-concepts-2408080.pdf

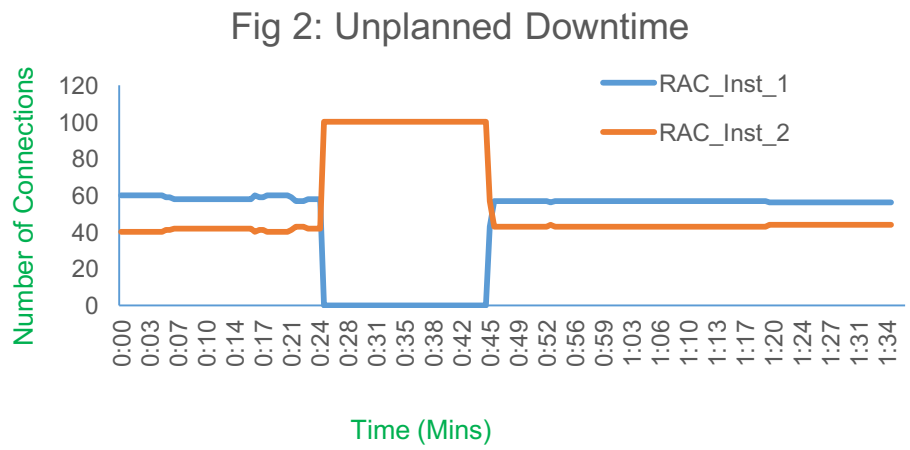
Make sure AC is enabled on the server side to achieve continuous service without any interruption of in-flight work. AC invokes TG internally to reliably determine the outcome of the transaction before replaying it. Please refer to the white paper **Java Programming with Oracle Database 12c RAC and Active Data Guard**[7] for understanding how TG and AC will protect your application from unplanned downtimes.

**DBA or RDBMS Steps**

To simulate Fast Connection Failover, the DBA may either stop the service on one instance with –force option (as specified hereafter) or, alternatively, kill the Oracle instance SMON background process. An even more drastic approach consists in powering down of one of the nodes supporting the database.

```
$srvctl stop service -db <db_name> -service <service_name> -instance <instance_name> -
force
```

Fig 2, shows connections distribution of XYZ service across two RAC instances before and after unplanned downtime. Notice that the connection workload went from fifty-fifty across both instances to hundred-zero. In other words, the remaining instances sustain the workload without disrupting the business operation.




**Application Continuity Checklist**

(1) **Always use the recommended connection URL with connection descriptors for HA.**
Take note of these best practices while forming a connection URL.

- DO NOT use EasyConnection URL (Example: *jdbc:oracle:thin:hr/hr@localhost:5221:orcl*) does not provide any HA capabilities.

7 http://www-content.oracle.com/technetwork/database/application-development/12c-ha-concepts-2408080.pdf

- Always use the long form connection URL with descriptors CONNECT_TIMEOUT, RETRY_COUNT, RETRY_DELAY, TRANSPORT_CONNECT_TIMEOUT as these connection descriptors allow requests to wait for the service and connect successfully
    - Use one DESCRIPTION and more than one causes long delays
    - Set LOAD_BALANCE= ON per ADDRESS_LIST to balance SCANs
    - DO NOT use RETRY_COUNT without RETRY_DELAY
    - Set CONNECT_TIMEOUT to a high value to prevent logon storms.

Sample Connection URL:

```
alias =(DESCRIPTION =
          CONNECT_TIMEOUT=120) (RETRY_COUNT=20) (RETRY_DELAY=3)
(TRANSPORT_CONNECT_TIMEOUT=3)
       (ADDRESS_LIST =(LOAD_BALANCE=on)
         (ADDRESS = (PROTOCOL = TCP) (HOST=primary-scan) (PORT=1521)))
       (ADDRESS_LIST =(LOAD_BALANCE=on)
         (ADDRESS = (PROTOCOL = TCP) (HOST=secondary-scan) (PORT=1521)))
 (CONNECT_DATA=(SERVICE_NAME = gold-cloud)))
```

(2) **Enable JDBC Statement Cache**

The recommendation is to use JDBC statement cache and disable a statement cache at Tomcat. JDBC statement cache is optimized and supports Application Continuity.

```
oracle.jdbc.implicitstatementcachesize=nnn (nnn = number of statements to be cached)
OracleDataSource ods = new OracleDataSource();
ods.setImplicitCachingEnabled(true);
```

(3) **Align Application and Server Timeouts**

If the application timeout (*http_request_timeout*) is lower than failover timeout (READ_TIMEOUT set at the connection level) then replays will cancel and repeat or application receives timeout errors. So, make sure *http_request_timeout* is higher than READ_TIMEOUT.

(4) **Disable AUTOCOMMIT**

Use the UCP property autoCommit = false. Refer to *Fig 1a: Database Resource in Tomcat with 12.2 UCP* for more details.

(5) **Always return connections to the pool**

The best practice is that an application checks-out a connection only for the time that it needs it. Holding a connection when it is not in use is not a good practice. Therefore, an application should check-in the connection back when the work is completed. This way, connections are available for subsequent use by other threads when required.

(6) **JDBC concrete classes are not supported**

Run Orachk to find out if the application is using concrete classes. Replace these concrete classes before AC takes effect. Refer to MOS note 1364193.1 for using the new datatypes while replacing the concrete classes.
Example., oracle.sql.CLOB, oracle.sql.BLOB, oracle.sql.ARRAY etc., are concrete classes.

(7) **Use APIs to check the AC statistics**

There are two APIs `OracleDataSource.getReplayStatistics(StatisticsReportType)` and `OracleDataSource.getReplayStatistics()` that provide some metrics around the AC protection. AC statistics such as total number of requests, number of completed requests, number of successful replay requests, number of failed replay requests, number of disabled replay requests, number of failed attempts etc., can be analyzed through these APIs.

(8) **Tune garbage collector**

For many applications the default garbage collector tuning is correct. But, for an application with extreme performance requirements, it may be necessary to perform JVM tuning. Use the attributes below for such scenarios. (Note that both attributes should be set to the same value).

*-Xms200m –Xmx200m*

## Runtime Load Balancing (RLB) with Tomcat Servlets

Runtime Connection Load Balancing enables routing of work requests across RAC or ADG instances to achieve predictable runtime performance. RAC and GDS post runtime load balancing advisories every 30 seconds. UCP uses the load balancing advisory to balance the work across RAC instances, dynamically and thereby achieving best scalability. Runtime Load Balancing comes also into play when new node(s)/instance(s) are added/removed to/from the service; the work load gets balanced in both situations without any manual intervention.

### Web Application steps

FCF is required to allow receiving FAN load balancing advisories. Refer to "*Hiding Planned Maintenance from Tomcat Servlets"* for more details on enabling FCF. UCP dispenses connections from the least loaded database instance (in RAC or GDS environments). Ultimately the workload is uniformly spread across the databases in question (RAC or GDS).

### DBA or RDBMS Steps

Configure the Oracle RAC Load Balancing Advisory with the following values.

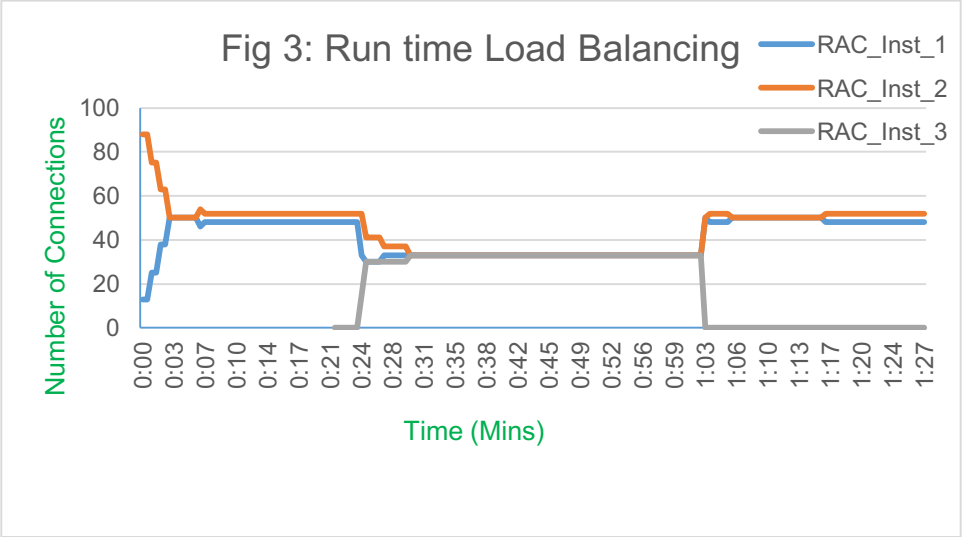
Set 'Runtime Load Balancing Goal' to SERVICE_TIME or THROUGHPUT

```
$srvctl modify service –db <db_name> -service <service_name> -rlbgoal SERVICE_TIME
$gdsctl modify service –db <db_name> -service <service_name> -rlbgoal SERVICE_TIME
```

Set 'Connection Load Balancing Goal' to SHORT

```
$srvctl modify service –db <db_name> -service <service_name> -clbgoal SHORT
$gdsctl modify service –db <db_name> -service <service_name> -clbgoal SHORT
```

Fig 3, shows connections distribution of XYZ service across three RAC instances. Notice that the workload is gradually distributed across the available instances with 48-51 connections each between RAC_Instance_1 and RAC_Instance_2. When a new instance, RAC_Instance_3 is added, the load will be re-distributed evenly to 33-33-33. After some time, RAC_Instance_3 is removed, UCP gradually rebalances the load between the remaining instances and in this case, achieves 48-51 connection workload distribution.




## Appendix

**Enable JDBC & UCP logging for debugging**

Enable JDBC & UCP logging when there are issues. This helps to debug and find the root cause of the problem. There are few steps for enabling JDBC & UCP logging.

- Configure debug jar in the classpath
- Enable logging
- Setup a config file for advanced logging

**Configure debug jar in the classpath:**
Make sure to have the debug jar file **ojdbc8_g.jar** in the classpath.

**Enable logging**

In order to get any log output from the Oracle JDBC drivers you must enable logging. Enable logging by setting the system property -D**oracle.jdbc.Trace = TRUE**. This turns logging ON. Refer to *Fig 5: Enable JDBC/UCP logging in Tomcat.*

**Setup a config file for advanced logging**

Create a configuration file, for example *oracletrace.properties* and insert the following and save the file.
Enable the config file by setting the system property ***–Djava.util.logging.config.file =<location of the config file>.***
Refer to *Fig 5: Enable JDBC/UCP Logging in Tomcat.*

```
# FOR UCP logs
.level=WARNING
oracle.ucp.jdbc.oracle.level=FINEST
oracle.ucp.jdbc.level=FINEST
oracle.ucp.common.level=FINEST
oracle.ucp.jdbc.oracle.rlb.level=FINEST
```

```
# For JDBC Driver logs
level=SEVERE
oracle.jdbc.level=ALL
oracle.jdbc.driver.level=FINEST
oracle.jdbc.pool.level=FINEST
oracle.jdbc.util.level=OFF
oracle.jdbc.handlers=java.util.logging.FileHandler
java.util.logging.FileHandler.level=FINE
java.util.logging.FileHandler.pattern=jdbc.log
java.util.logging.FileHandler.count=1
java.util.logging.FileHandler.formatter=java.util.logging.SimpleFormatter
```

**Fig 5: Enabling JDBC/UCP logging in Tomcat**

```
<!—System property for graceful planned maintenance -->
<context-param>
<param-name>oracle.jdbc.Trace</param-name>
<param-value>true</param-value>
<param-name>java.util.logging.config.file</param-name>
<param-value>/opt/tomcat/logs/oracletrace/oracletrace.properties</param-value>
</context-param>
```

## Conclusion

This paper gives you a comprehensive and practical coverage of Tomcat Web applications with Oracle Database 12c; more specifically how to design and configure both the RDBMS, UCP, and the Tomcat container for resiliency to planned, and unplanned database downtimes and workload balancing. The steps described in this paper are valid for all Oracle Database 12c high availability and scalability configurations including RAC, Active Data Guard, and Global Data Services. The complete UCP Tomcat demo referenced in this paper is posted on https://github.com/oracle/oracle-db-examples/tree/master/java. Java architects, Web application designers and DBAs may now design robust and reliable Tomcat Web applications for better user experience and operation continuity.

**Oracle Corporation, World Headquarters**
500 Oracle Parkway
Redwood Shores, CA 94065, USA

**Worldwide Inquiries**
Phone: +1.650.506.7000
Fax: +1.650.506.7200

ORACLE

CONNECT WITH B

blogs.oracle.com
facebook.com/oracle
twitter.com/oracle
oracle.com

Hardware and Software, Engineered to Work Together