

Oracle Maximum  
Availability Architecture

# Client Failover Best Practices for Highly Available Oracle Databases

Oracle Database 12c

ORACLE WHITE PAPER | AUGUST 2015





## Table of Contents

Overview	1
Introduction	2
Automatic Failover for Client Applications that Support FAN	3
Prerequisites for Automating Client Failover – JDBC, OCI and ODP.net	4
Role-Based Database Services	5
Configuring Automatic Failover for JDBC Clients	6
Configuring Application Continuity and Transaction Guard	8
Configuring OCI Clients for Automatic Failover	10
Configuring Automatic Failover for ODP.Net Clients	13
Configuring Oracle Database Services:	14
Role Transition Timings	15
Automatic Failover for Clients that do not Support FAN	18
Additional Considerations for Active Data Guard Connections	18
Controlling Logon Storms	20
Conclusion	20

## Overview

The Oracle Maximum Availability Architecture (MAA)<sup>1</sup> is the best practices blueprint for implementing Oracle high availability technologies. The core principle of high availability is to make outages as transparent as possible to the user. Oracle MAA with Oracle Database 12c is unique in enabling users to continue processing transactions with zero interruption (applications do not receive an error and in-flight transactions succeed) through component, system, and database outages or during planned maintenance. Oracle Real Application Cluster (Oracle RAC), Oracle Data Guard, Oracle Active Data Guard, and Application Continuity are key components of Oracle MAA that enable this capability.

- » Oracle RAC enables multiple active Oracle instances running on separate servers to share access to a single Oracle Database, providing both high availability (HA) and scalability.
- » Data Guard and Active Data Guard provide the management, monitoring, and automation software to create and maintain one or more synchronized physical replicas (standby databases), of a single instance or Oracle RAC production database (the primary database), to protect Oracle data in the event of system or cluster failures, disasters, errors, data corruptions, or other outages. If a primary database becomes unavailable for any reason, Data Guard and Active Data Guard can execute an automatic failover to a standby database in order to maintain high availability.
- » Oracle MAA best practices and the event notification framework of Oracle Database 12c automatically transition applications connected to the Oracle Database from one RAC node to another in the event of a server or database instance outage, or from a primary database to a standby database in the event of a database outage. This same framework is also used during planned maintenance – RAC rolling maintenance across instances in an Oracle RAC database and Data Guard rolling maintenance across primary and standby databases.
- » Application Continuity with Transaction Guard safely replays in-flight uncommitted transactions to mask the outage of an Oracle instance or database from the user. Applications do not receive errors and users do not need to resubmit transactions.

This technical white paper is intended for database administrators and developers. It describes Oracle Database 12c configuration best practices to automatically transition application connections from a failed primary database to a new primary database after a Data Guard / Active Data Guard role transition has occurred. This paper also describes best practices for Application Continuity and Transaction Guard, new with Oracle Database 12c.

---

<sup>1</sup> <http://www.oracle.com/goto/maa>

## Introduction

Unplanned failures of an Oracle Database instance in a high availability configuration fall into three general categories:

### Server or Instance Failure in an Oracle RAC Cluster

When an outage occurs due to server failure or other fault that causes the crash of an individual Oracle instance or database server in an Oracle RAC cluster, availability is restored by rapidly notifying clients connected to the failed instance and immediately reconnecting them to surviving instances in the cluster. Fast Application Notification (FAN) will break connected clients out of TCP timeout, and Transparent Application Failover (OCI clients) or Fast Connection Failover (JDBC clients) will automatically fail clients over to database services running on surviving database instances. Detailed best practices for this category of failure are described in *Oracle Real Application Clusters Administration and Deployment Guide 12c Release 1 (12.1), Chapter 5 Workload Management with Dynamic Database Services*<sup>2</sup> and additional information in *My Oracle Support Note 1593712.1, Graceful Application Switchover in RAC with No Application Interruption*<sup>3</sup>.

New Oracle Database 12c features with Oracle RAC include:

- » Transaction Guard is an API used by applications to determine the last outcome of a failed session and transaction
- » Application Continuity is a solution that masks application outages by rebuilding the database session, and resubmitting the pending work following recoverable errors that make the database session unavailable.

Best practices for the use of Application Continuity with Oracle RAC are described in the whitepaper, *Application Continuity with Oracle Database 12c*<sup>4</sup>

### Complete Site Failure

A complete-site failure results in both the application and database tiers being unavailable. To maintain availability users must be redirected to a secondary site that hosts a redundant application tier and a synchronized copy of the production database. MAA best practice is to maintain a running application tier at the standby site to avoid startup time and use Data Guard to maintain the synchronized copy of the production database. An example of such a configuration is provided in the paper, *Best Practices for Oracle Fusion Middleware SOA Multi Data Center Active-Active Deployment*<sup>5</sup>.

Upon site failure a WAN traffic manager is used to execute a DNS failover (either manually or automatically) to redirect all users to the application tier at standby site while a Data Guard failover transitions the standby database to the primary production role. See the *Oracle Database High Availability Best Practices*<sup>6</sup> documentation for information about automating complete site failover.

<sup>2</sup> [http://docs.oracle.com/cd/E16655\\_01/rac.121/e17887/hafeats.htm#RACAD076](http://docs.oracle.com/cd/E16655_01/rac.121/e17887/hafeats.htm#RACAD076)

<sup>3</sup> <https://support.oracle.com/CSP/main/article?cmd=show&type=NOT&id=1593712.1>

<sup>4</sup> <http://www.oracle.com/technetwork/database/database-cloud/private/application-continuity-wp-12c-1966213.pdf>

<sup>5</sup> <http://www.oracle.com/technetwork/database/availability/fmw11gsoamultidc-aa-1998491.pdf>

<sup>6</sup> [http://download.oracle.com/docs/cd/B19306\\_01/server.102/b25159/outage.htm#BABHCAIA](http://download.oracle.com/docs/cd/B19306_01/server.102/b25159/outage.htm#BABHCAIA)

## Partial Site Failure

A partial-site failure is when the primary database (a single-instance database or all nodes in an Oracle RAC database) becomes unavailable but the application tier that was connected to the primary database remains intact.

All that is required to maintain availability is to redirect the application tier to the new primary database after a Data Guard failover has completed. In this use-case the standby database is located within a distance from the surviving application tier where it can deliver acceptable performance using a remote connection after a database failover has occurred.

Similar to the Oracle RAC use case, FAN will break connected clients out of TCP timeout, and Transparent Application Failover (OCI clients) or Fast Connection Failover (JDBC clients) will automatically fail applications over to the new primary database. This process applies to Data Guard configurations having Oracle RAC databases or to single-instance, non-RAC databases that use Oracle Restart<sup>7</sup>. Similar to Oracle RAC, all of the benefits of Application Continuity and Transaction Guard also apply to Data Guard switchovers (planned events) and to zero data loss failover (an unplanned outage) when Data Guard automatic failover (Fast-Start Failover) is used.

This remainder of this paper provides configuration best practices for automatic application failover to a new primary database for this category of outage.

---

*Note: Unless specifically noted, all references to Data Guard in this paper refer collectively to both Data Guard (features included with Oracle Database Enterprise Edition) and Active Data Guard (advanced capabilities included in an option license for Oracle Enterprise Edition)*

---

## Automatic Failover for Client Applications that Support FAN

At a high level, automating client failover in a Data Guard configuration includes relocating database services to the new primary database as part of a Data Guard failover, notifying clients that a failure has occurred to break them out of TCP timeout, and redirecting clients to the new primary database.

This paper describes how to create role-based database services in a Data Guard configuration. It includes detailed configuration steps for enabling OCI and JDBC, OLE DB, and ODP .Net application clients to receive FAN notifications and quickly reconnect to a new primary database.

Packaged middleware applications, such as Weblogic, Fusion Middleware, etc, utilize the same basic configuration described in this paper with some additional application specific configuration. These configuration details for packaged applications are outside the scope of this paper. For more information see:

- » *Client and Application Failover Validation Matrix, My Oracle Support Note 1617163.*<sup>8</sup>
- » *Oracle WebLogic Server and Oracle Database: Oracle Integrated Maximum Availability Solutions*<sup>9</sup>
- » *Application Continuity with Oracle Database 12c*<sup>10</sup>

If your application client does not support FAN, then see the section of this paper titled "[Automatic Failover for Clients Applications That Do Not Support FAN.](#)"

<sup>7</sup> <http://docs.oracle.com/database/121/ADMIN/restart.htm#ADMIN12708>

<sup>8</sup> <https://support.oracle.com/CSP/main/article?cmd=show&type=NOT&id=1617163.1>

<sup>9</sup> <http://www.oracle.com/technetwork/database/features/availability/wlsdatasourcefordataguard-1534212.pdf>

<sup>10</sup> <http://www.oracle.com/technetwork/database/database-cloud/private/application-continuity-wp-12c-1966213.pdf>

## Prerequisites for Automating Client Failover – JDBC, OCI and ODP.net

The best practices described in this paper require Oracle Database 12.1.0.2 or later.

Database Services are foundational to the application failover best practices described in this paper. If you do not already have a thorough understanding of database services please review *Oracle Real Application Clusters Administration and Deployment Guide 12c Release 1 (12.1) guide Chapter 5 Workload Management with Dynamic Database Services*<sup>11</sup> before proceeding. For single instance database environments (non-RAC) refer to Oracle Restart<sup>12</sup> documentation for an understanding of how to manage services.

---

*Note: the Oracle Database 12c documentation for Restart includes a notice that this feature is being deprecated. At the time this paper was written there was no plan to discontinue support or bug fixes for Restart until a minimum of one release after a superseding feature has been released. This means customers requiring the functionality of Restart may continue to deploy it with confidence even though no superseding feature has been announced. For the latest information regarding the status of Oracle Restart see My Oracle Support Note 1584742.1)*

---

The best practices described in this paper require that the Data Guard configuration be managed by the Data Guard Broker<sup>13</sup>. The Data Guard Broker is responsible for sending FAN events to client applications in order to clean up their connections to the down database and reconnect them to the new production database. The Data Guard broker also coordinates with Oracle Clusterware (or Oracle Restart for single instance databases) to properly fail over role-based services to a new primary database after a Data Guard failover has occurred.

Data Guard Fast-Start Failover (FSFO)<sup>14</sup> is recommended for complete automation of the database failover process. FSFO eliminates the requirement for human intervention to execute a failover should the primary database become unavailable. FSFO is a tightly controlled process that ensures data protection and availability objectives are met. FSFO is also required to use Application Continuity for failover to a standby database.

---

*Note: Global Data Services (GDS)<sup>15</sup>, new with Oracle Database 12c, provides a global mechanism for workload load balancing and database service management for replicated databases, including Data Guard and Oracle GoldenGate. While GDS utilizes the same event notification framework as Oracle RAC and Data Guard, it provides its own methods for event handling and service management - it does not utilize the Data Guard Broker, Oracle Clusterware, or Oracle Restart as described in this paper for these purposes. While GDS has much in common with the best practices described in this paper, GDS represents an alternative approach to automating client failover and is outside the scope of this paper.*

---

<sup>11</sup> [http://docs.oracle.com/cd/E16655\\_01/rac.121/e17887/hafeats.htm#RACAD076](http://docs.oracle.com/cd/E16655_01/rac.121/e17887/hafeats.htm#RACAD076)

<sup>12</sup> [http://docs.oracle.com/cd/E16655\\_01/server.121/e17636/restart.htm#ADMIN12709](http://docs.oracle.com/cd/E16655_01/server.121/e17636/restart.htm#ADMIN12709)

<sup>13</sup> [http://docs.oracle.com/cd/E16655\\_01/server.121/e17641/toc.htm](http://docs.oracle.com/cd/E16655_01/server.121/e17641/toc.htm)

<sup>14</sup> <http://docs.oracle.com/database/121/DGBKR/sofo.htm#i1027843>

<sup>15</sup> <http://www.oracle.com/technetwork/database/features/availability/global-data-services-1949717.html>

To receive and react to FAN events, client applications and connection pools must meet the following requirements:

#### JDBC applications:

- » The implicit connection cache is enabled.
- » The application uses service names to connect to the database.
- » The underlying database is an Oracle RAC or has Oracle Restart capability (for single-instance databases)
- » Oracle Notification Service (ONS) is configured and available on the node where JDBC is running.
- » The Java Virtual Machine (JVM) in which your JDBC instance is running must have `oracle.ons.oraclehome` set to point to your `ORACLE_HOME`.

For more information, see the Oracle Universal Connection Pool for JDBC Developer's Guide<sup>16</sup>.

#### OCI applications:

- » An Oracle RAC environment with Oracle Clusterware set up and enabled or a single node (non-Oracle RAC) database with Oracle Restart.
- » The application must have been linked with the threads library.
- » The OCI environment must be created in `OCI_EVENTS` and `OCI_THREADED` mode.

For more information, see the Oracle Call Interface Programmer's Guide<sup>17</sup>.

#### ODP.Net:

- » Namespace: `Oracle.DataAccess.Client`, Assembly: `Oracle.DataAccess.dll`.
- » Microsoft .NET Framework Version 2.0 or later.

For more information, see the Oracle Database Administrator's Guide<sup>18</sup>.

## Role-Based Database Services

Beginning with Data Guard 11g Release 2 (11.2), you can automatically control the startup of database services on primary and standby databases by assigning a database role [-I `[[PRIMARY] | [PHYSICAL_STANDBY] | [LOGICAL_STANDBY] | [SNAPSHOT_STANDBY]]`] to each service<sup>19</sup>. A database service automatically starts upon database startup if the management policy of the service is `AUTOMATIC` and if one of the roles assigned to that service matches the current role of the database.

Services must be configured with the Server Control (`SRVCTL`) utility identically on all databases in a Data Guard configuration. In the examples used in the following sections, a service named `oltpworkload` is configured to be active when the database Austin is in the primary role (-I `PRIMARY`). The same service is also configured on the standby database Houston so that is started whenever Houston functions in the primary role.

Similarly, a second service named `reports` is configured to be started when Austin or Houston are functioning in the standby database role (-I `PHYSICAL_STANDBY`). The `reports` service provides real-time reporting using Active Data Guard (the standby database is open read-only at the same time it is applying redo received from the primary database).

<sup>16</sup> [http://docs.oracle.com/cd/E16655\\_01/java.121/e17659/rac.htm#JJUCP08100](http://docs.oracle.com/cd/E16655_01/java.121/e17659/rac.htm#JJUCP08100)

<sup>17</sup> [http://download.oracle.com/docs/cd/E11882\\_01/appdev.112/e10646/oci09adv.htm#sthref1523](http://download.oracle.com/docs/cd/E11882_01/appdev.112/e10646/oci09adv.htm#sthref1523)

<sup>18</sup> [http://download.oracle.com/docs/cd/E11882\\_01/server.112/e17120/restart002.htm#ADMIN13196](http://download.oracle.com/docs/cd/E11882_01/server.112/e17120/restart002.htm#ADMIN13196)

<sup>19</sup> [http://download.oracle.com/docs/cd/E11882\\_01/rac.112/e16795/hafeats.htm#RACAD7126](http://download.oracle.com/docs/cd/E11882_01/rac.112/e16795/hafeats.htm#RACAD7126)

## Configuring Automatic Failover for JDBC Clients

### Prerequisites:

- » The Universal Connection Pool (UCP) is enabled (UCP 12.1.0.2 or later).
- » The application uses service names to connect to the database.
- » Oracle Notification Service (ONS) is configured and available on the node where JDBC is running.
- » The Java Virtual Machine (JVM) in which your JDBC instance is running must have `oracle.ons.oraclehome` set to point to your `ORACLE_HOME`.

### Configuration:

1. Enable Fast Connection Failover (FCF) and configure the JDBC application to connect to all ONS daemons for both the primary and standby clusters using the `setONSConfiguration` property. The `setONSConfiguration` property should point to all primary and standby ONS daemons.

```
pds.setONSConfiguration("nodes=adczatdb01:6200,adczatdb02:6200,slcc17adm01:6200,slc  
c17adm02:6200");  
pds.setFastConnectionFailoverEnabled(true);
```

2. By default the JDBC application will randomly pick three hosts from the `setONSConfiguration` property and create connections to those three ONS daemons. This default must be changed so that connections are made to all ONS daemons. This is done by setting the following property when the JDBC application is invoked to the total number of ONS daemons in the configuration:

```
java -Doracle.ons.maxconnections=4
```

3. The JDBC client must set the `oracle.net.ns.SQLnetDef.TCP_CONNTIMEOUT_STR` property. This property enables the JDBC client to quickly traverse an `ADDRESS_LIST` in the event of a failure. For example, if the client attempts to connect to a host that is unavailable, the connection attempt will be bounded to the time specified by the `SQLnetDef.TCP_CONNTIMEOUT_STR` property after which the client attempts to connect to the next host in the `ADDRESS_LIST`. The behavior continues for each host in the `ADDRESS_LIST` until a connection is made. Setting the property to a value of 3 seconds will suffice in most environments. It is important to note that the `SQLnetDef.TCP_CONNTIMEOUT_STR` property should be set on the data source and not on the Universal Connection Pool.

```
Properties prop = new Properties();  
prop.put(oracle.net.ns.SQLnetDef.TCP_CONNTIMEOUT_STR, "+3000"); // 3000ms  
pds.setConnectionProperties(prop);
```

4. Set the following to get the correct behavior from SCAN load balancing:

```
// need to set oracle.jdbc.thinForceDNSLoadBalancing  
prop.put("oracle.jdbc.thinForceDNSLoadBalancing", "true");
```

5. Configure JDBC clients to use a connect descriptor that includes an address list that in turn includes the SCAN address for each site and connects to an existing service. Do not configure both TAF and JDBC FCF when using JDBC thick clients.



The following URL searches both primary and standby sites looking for the appropriate service with very little overhead. This URL configuration is the recommended approach:

```
PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
pds.setUser("system");
pds.setPassword("oracle");
String dbURL =
"jdbc:oracle:thin:@" +
"(DESCRIPTION=" +
"(FAILOVER=on)" +
"(ADDRESS_LIST=" +
"(LOAD_BALANCE=on)" +
"(CONNECT_TIMEOUT=3)(RETRY_COUNT=3)" +
"(ADDRESS=(PROTOCOL=TCP)(HOST=prmy-scan)(PORT=1521))"+
"(ADDRESS=(PROTOCOL=TCP)(HOST= stby-scan)(PORT=1521)))" +
"(CONNECT_DATA=(SERVICE_NAME=oltpworkload))"
System.out.println("Url=" + dbURL);
pds.setURL(dbURL);
```

The following URL should be used if it is very rare for the primary to ever run on the secondary site and you wish to have connections connect as fast as possible:

```
PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
pds.setUser("system");
pds.setPassword("oracle");
String dbURL =
"jdbc:oracle:thin:@" +
"(DESCRIPTION_LIST=" +
"(LOAD_BALANCE=off)" +
"(FAILOVER=on)" +
"(DESCRIPTION=" +
"(CONNECT_TIMEOUT=3)(RETRY_COUNT=3)" +
"(ADDRESS_LIST=" +
"(LOAD_BALANCE=on)" +
"(ADDRESS=(PROTOCOL=TCP)(HOST=prmy-scan)(PORT=1521)))" +
"(CONNECT_DATA=(SERVICE_NAME=oltpworkload))" +
"(DESCRIPTION=" +
"(ADDRESS_LIST=" +
"(LOAD_BALANCE=on)" +
"(ADDRESS=(PROTOCOL=TCP)(HOST= stby-scan)(PORT=1521)))" +
"(CONNECT_DATA=(SERVICE_NAME=oltpworkload))");
System.out.println("Url=" + dbURL);
pds.setURL(dbURL);
```

Note that if a switchover or failover occurs the above URL will force all connections to go through the old prmy-scan before using the stby-scan where the primary currently runs.

When a new connection is made using the above URL's the following logic is used:

- » Oracle Net contacts DNS and resolves prmy-scan to a total of three IP addresses.
- » Oracle Net randomly picks one of the three IP address and attempts to make a connection. All three IP addresses will be tried a total of four times (initial attempt plus RETRY\_COUNT in the above example).
- » If the connection to primary site is unsuccessful, it then contacts DNS and resolves stby-scan to three addresses.
- » The same sequence is performed for the standby stby-scan as it was for the prmy-scan.

The following list provides additional information about the Oracle Net parameters used in the above alias:

- » `LOAD_BALANCE` is ON by default for `DESCRIPTION_LIST` only. This parameter by default is OFF for an address list within a `DESCRIPTION`. Setting this ON for a SCAN-based address implies that new connections will be randomly assigned to one of the 3 SCAN-based IP addresses resolved by DNS.
- » The default value for the `FAILOVER` parameter is ON for an address list within a `DESCRIPTION`. This impacts the 3 SCAN IP addresses the same way as if those 3 IP addresses were listed explicitly in the connect descriptor. This means that if the initial connection requests to the first randomly-assigned SCAN IP address fails, the connection will failover to another SCAN IP address, and will continue to do so, till it iterates the complete address list. Note that this parameter is relevant only to new connections. Failover of existing connections is handled by TAF, which is controlled by the separate `FAILOVER_MODE` parameter.
- » The `CONNECT_TIMEOUT` parameter is the time to connect to the database instance providing the requested service, and includes the time to establish a TCP connection to the listener. The timeout interval is applicable for each `ADDRESS` in an `ADDRESS_LIST`, and each IP address to which a host name is mapped. Set the `CONNECT_TIMEOUT` parameter to the maximum amount of time (in seconds) to wait for a response from an address before skipping to the next address. A setting of three seconds is recommended and is acceptable in most cases. Do not set this parameter to fewer than three seconds.

The equivalent global parameter in `sqlnet.ora` is `SQLNET.OUTBOUND_CONNECT_TIMEOUT`. If the same timeout value is sufficient for all connect strings, it would be simpler to set the global parameter. Otherwise, a separate setting can be done for each connect string.

- » The `RETRY_COUNT` parameter specifies the number of times an address list is traversed before the new connection attempt is terminated. The default value is 0. With respect to SCAN, with `FAILOVER = on`, setting this `RETRY_COUNT` parameter to a value of 2, for example, means the three SCAN IP addresses are traversed thrice (i.e.  $3*3=9$  connect attempts), before the connection is terminated:
- » When the connection request initially comes in, the first randomly assigned IP address tries to service that request, followed by the two remaining IP addresses. (This behavior is controlled by the `FAILOVER` parameter.)
- » The retries then kick in and the list of three IP addresses is tried two more times. `RETRY_COUNT` is only supported at `DESCRIPTION` level in connect string, but not at global (i.e. `sqlnet.ora`) level.

## Configuring Application Continuity and Transaction Guard

Prior to Application Continuity, the responsibility to detect, mask and react to any errors was left to the developer. This was often difficult to implement and could result in errors, timeouts and lost productivity for the overall application. Application Continuity removes this responsibility from the application developer by automatically masking outage from end users and application by recovering in-flight transactions and requests. All recovery is performed transparent to the application so that the end users only see a slight delay in execution. This is true for both planned and unplanned outages.

Application Continuity with Oracle Database 12c is used to mask outages for planned maintenance that is performed in rolling fashion across Oracle RAC instances or across a Data Guard primary and standby database. Application Continuity also masks unplanned outages of an Oracle RAC instance or a Data Guard primary database configured

in Maximum Availability (zero data loss failover) with Data Guard Fast-Start Failover (automatic database failover). Use of Application Continuity for a Data Guard failover requires that both source and target databases be at Oracle 12.1.0.2 or later.

Application Continuity is available for:

- » Oracle JDBC Replay Driver 12c or later. This is a JDBC driver feature provided with Oracle Database 12c for Application Continuity, referred to as the “replay driver” onwards (OCI support is planned for a future release).
- » Oracle Universal Connection Pool, Oracle WebLogic Server 12c (12.1.2) or later, and third-party Java connection pools or standalone Java applications – using Oracle JDBC- Replay Driver 12c or later.
- » Standard 3rd Party Java application servers using the pooled connection interface - including IBM WebSphere and Apache Tomcat, from 12.1.0.2
- » Standard 3rd Party Java application servers supporting the Universal Connection Pool as the pooled data source - including IBM WebSphere, Apache Tomcat, and RedHat JBoss
- » Third-party Java connection pools or standalone Java applications – using Oracle JDBC- Replay Driver 12c or later and embedding their own request boundaries

Application Continuity<sup>20</sup> uses Transaction Guard<sup>21</sup> to reliably determine if the last transaction was committed or not. Without Transaction Guard, applications and users who attempt to retry operations following an outage can cause logical corruption by committing duplicate transactions or committing transactions out of order.

Choose from one of the following options to configure Application Continuity using the Oracle JDBC 12c Replay Driver depending on your configuration:

#### Configure Oracle UCP 12c

- » Configure the Oracle JDBC 12c Replay Data Source as a connection factory on UCP PoolDataSource:  
`setConnectionFactoryClassName("oracle.jdbc.replay.OracleDataSourceImpl");`

#### Configure Oracle WebLogic Server 12c

- » Configure the Oracle 12c JDBC Replay Data Source using the Oracle WebLogic Server Administration Console

#### Configure Standalone Java Applications or Third-party Connection Pools

- » Configure the Oracle JDBC 12c Replay Data Source in the property file or in the thin JDBC application  
`replay_datasource=oracle.jdbc.replay.OracleDataSourceImpl`

#### Configure Connections for High Availability (failover and failback)

- » The REMOTE\_LISTENER setting for the database must include the addresses in the ADDRESS\_LISTs for all URL used for client connection:
  - » If any URL uses the SCAN Names, then REMOTE\_LISTENERS must include the SCAN Name.
  - » If any URL uses an ADDRESS\_LIST of host VIPs, then REMOTE\_LISTENERS must include an ADDRESS list including all SCAN VIPs and all host VIPs
- » Set RETRY\_COUNT, CONNECT\_TIMEOUT parameters in the URL to allow new incoming connections to retry. For a complete discussion on this attributes please consult the JDBC Application Configuration Section later in this paper.

Example:

```
"jdbc:oracle:thin:@" +
```

<sup>20</sup> <http://www.oracle.com/technetwork/database/database-cloud/private/application-continuity-wp-12c-1966213.pdf>

<sup>21</sup> <http://www.oracle.com/technetwork/database/database-cloud/private/transaction-guard-wp-12c-1966209.pdf>

```

" (DESCRIPTION=" +
"(FAILOVER=on)" +
" (ADDRESS_LIST=" +
" (LOAD_BALANCE=on)" +
" (CONNECT_TIMEOUT=3) (RETRY_COUNT=3)" +
" (ADDRESS=(PROTOCOL=TCP) (HOST=prmy-scan) (PORT=1521)) "+
" (ADDRESS=(PROTOCOL=TCP) (HOST= stby-scan) (PORT=1521))" +
" (CONNECT_DATA=(SERVICE_NAME=oltpworkload))"

```

### Configure Services for Application Continuity

- » Set the service attributes using SRVCTL / GDSCTL to use Application Continuity,
- » Set FAILOVER\_TYPE to TRANSACTION to enable Application Continuity
- » Set COMMIT\_OUTCOME to TRUE to enable Transaction Guard (mandatory)
- » Also review the following service attributes:
  - » REPLAY\_INITIATION\_TIMEOUT : Set this to the duration in seconds after which replay is not started (e.g. 180, 300, 1800 seconds – the override to cancel replay). This timer starts at beginRequest. (default 300 seconds)
  - » FAILOVER\_RETRIES : Set this to specify the number of connection retries for each replay attempt. (default 30 retries, applied at replay driver)
  - » FAILOVER\_DELAY : Set this to specify the delay in seconds between connection retries (default 10 seconds, applied at replay driver)
  - » AQ\_HA\_NOTIFICATIONS: Set this to TRUE to enable FAN (default TRUE)
  - » Example

```

srvctl add service -db mts -service oltpworkload -role PRIMARY -notification
TRUE -session_state dynamic -failovermethod basic -failovermethod basic -
commit_outcome TRUE -failoverretry 30 -failoverdelay 10 -replay_init_time 900 -
clbgoal SHORT -rlbgoal SERVICE_TIME -preferred mts1,mts2 -retention 3600 -
verbose

```

### Check Resource Allocation

- » Ensure that the system has the necessary memory and CPU resources.
- » Memory: The JDBC replay driver uses more memory than the base JDBC driver because the calls are retained until the end of a database request. If the number of calls retained is small, then the memory consumption of the replay driver is comparable to the base driver. At the end of a request, the calls are released to the garbage collector. This action differs from the base driver that releases as calls are closed.
- » For good performance, if there is sufficient memory, allocate 4 to 8 GB (or more) of memory for the Virtual Machine (VM), for example, by setting -Xms4096m for 4 GB.
- » CPU: The JDBC replay driver uses some additional CPU for building proxy objects, managing queues, and for garbage collection. The server uses some additional CPU for managing the validation. CPU overhead is red.

## Configuring OCI Clients for Automatic Failover

### Prerequisites

- » An Oracle RAC environment with Oracle Clusterware set up and enabled or a single node (non-Oracle RAC) database with Oracle Restart.
- » The application must have been linked with the threads library.

» The OCI environment must be created in OCI\_EVENTS and OCI\_THREADED mode.

## Configuration

Enable FAN for OCI clients by initializing the environment with the OCI\_EVENTS parameter, as follows:

OCIEnvCreate(...OCI\_EVENTS...)

1. Link the OCI client applications with thread library libthread or libpthread
2. Enable the application to check if an event has occurred by using code similar to the following example:

```
void evtcallback_fn(ha_ctx, eventhp)
...
printf("HA Event received.\n");
if (OCIHandleAlloc( (dvoid *)envhp, (dvoid **)&errhp, (ub4) OCI_HTYPE_ERROR,
                    (size_t) 0, (dvoid **) 0))
    return;
if (retcode = OCIAttrGet(eventhp, OCT_HTYPE_EVENT, (dvoid *)&srvhp, (ub4 *)0,
                        OCI_ATTR_HA_SRVFIRST, errhp))
    checkerr (errhp, (sword)retcode);
else {
    printf("found first server handle.\n");
    /*get associated instance name */
    if (retcode = OCIAttrGet(srvhp, OCI_HTYPE_SERVER, (dvoid *)&instname,
                            (ub4 *)&sizep, OCI_ATTR_INSTNAME, errhp))
        checkerr (errhp, (sword)retcode);
    else
        printf("instance name is %s.\n", instname);
}
```

3. Clients and applications can register a callback that is invoked whenever a high availability event occurs, as shown in the following example:

```
/*Registering HA callback function */
if (checkerr(errhp, OCIAttrSet(envhp, (ub4) OCI_HTYPE_ENV,
                              (dvoid *)evtcallback_fn, (ub4) 0,
                              (ub4)OCI_ATTR_EVTGBK, errhp)))
{
    printf("Failed to set register EVENT callback.\n");
    return EX_FAILURE;
}
if (checkerr(errhp, OCIAttrSet(envhp, (ub4) OCI_HTYPE_ENV,
                              (dvoid *)evtctx, (ub4) 0,
                              (ub4)OCI_ATTR_EVTCTX, errhp)))
{
    printf("Failed to set register EVENT callback context.\n");
    return EX_FAILURE;
}
```

```
return EX_SUCCESS;
```

After registering an event callback and context, OCI will call the registered function once for each high availability event.

4. Configure an Oracle Net alias that the OCI application will use to connect to the database. The Oracle Net alias should specify both the primary and standby SCAN hostnames. The following alias searches both primary and standby sites looking for the appropriate service with very little overhead. This alias configuration is the recommended approach:

```
SALES=
(DESCRIPTION=
(FAILOVER=on)
(CONNECT_TIMEOUT=5)(TRANSPORT_CONNECT_TIMEOUT=3)(RETRY_COUNT=3)
(ADDRESS_LIST=
(Load_Balance=on)
(ADDRESS=(PROTOCOL=TCP)(HOST=prmy-scan)(PORT=1521))
(ADDRESS=(PROTOCOL=TCP)(HOST=stby-scan)(PORT=1521)))
(CONNECT_DATA=(SERVICE_NAME=oltpworkload)))
```

The following alias should be used if it is very rare for the primary to ever run on the secondary site and you wish to have connections connect as fast as possible:

```
SALES=
(DESCRIPTION_LIST=
(Load_Balance=off)
(FAILOVER=on)
(DESCRIPTION=
(CONNECT_TIMEOUT=5)(TRANSPORT_CONNECT_TIMEOUT=3)(RETRY_COUNT=3)
(ADDRESS_LIST=
(Load_Balance=on)
(ADDRESS=(PROTOCOL=TCP)(HOST=prmy-scan)(PORT=1521)))
(CONNECT_DATA=(SERVICE_NAME=oltpworkload)))
(DESCRIPTION=
(CONNECT_TIMEOUT=5)(TRANSPORT_CONNECT_TIMEOUT=3)(RETRY_COUNT=3)
(ADDRESS_LIST=
(Load_Balance=on)
(ADDRESS=(PROTOCOL=TCP)(HOST=stby-scan)(PORT=1521)))
(CONNECT_DATA=(SERVICE_NAME=oltpworkload))))
```

When a new connection is made using the above Oracle Net alias the following logic is used:

- » Oracle Net contacts DNS and resolves prmy-scan to a total of three IP addresses.
- » Oracle Net randomly picks one of the three IP address and attempts to make a connection. If the connection attempt to the IP address does not respond in three seconds (TRANSPORT\_CONNECT\_TIMEOUT) the next IP address is attempted. All three IP addresses will be tried a total of four times (initial attempt plus RETRY\_COUNT in the above example).

- » If the connection to primary site is unsuccessful, it then contacts DNS and resolves stby-scan to three addresses.
- » The same sequence is performed for the standby stby-scan as it was for the prmy-scan.

The following list provides additional information about the Oracle Net parameters used in the above alias:

- » **LOAD\_BALANCE** is ON by default for **DESCRIPTION\_LIST** only. This parameter by default is OFF for an address list within a **DESCRIPTION**. Setting this ON for a SCAN-based address implies that new connections will be randomly assigned to one of the 3 SCAN-based IP addresses resolved by DNS.
- » The default value for the **FAILOVER** parameter is ON for an address list within a **DESCRIPTION**. This impacts the 3 SCAN IP addresses the same way as if those 3 IP addresses were listed explicitly in the connect descriptor. This means that if the initial connection requests to the first randomly-assigned SCAN IP address fails, the connection will failover to another SCAN IP address, and will continue to do so, till it iterates the complete address list. Note that this parameter is relevant only to new connections. Failover of existing connections is handled by TAF, which is controlled by the separate **FAILOVER\_MODE** parameter.

The **CONNECT\_TIMEOUT** parameter is the time to connect to the database instance providing the requested service, and includes the time to establish a TCP connection to the listener. The TCP duration is controlled by **TRANSPORT\_CONNECT\_TIMEOUT**, which has a default value of 60 seconds. If both timeouts are specified, it is recommended that **CONNECT\_TIMEOUT** be set to a value slightly greater than **TRANSPORT\_CONNECT\_TIMEOUT**. The timeout interval is applicable for each **ADDRESS** in an **ADDRESS\_LIST**, and each IP address to which a host name is mapped. Set the **CONNECT\_TIMEOUT** parameter to the maximum amount of time (in seconds) to wait for a response from an address before skipping to the next address. A setting of a minimum of three seconds is recommended and is acceptable in most cases. The equivalent global parameter in `sqlnet.ora` is `SQLNET.OUTBOUND_CONNECT_TIMEOUT`. If the same timeout value is sufficient for all connect strings, it would be simpler to set the global parameter. Otherwise, a separate setting can be done for each connect string.

The equivalent global parameter for **TRANSPORT\_CONNECT\_TIMEOUT** is `TCP.CONNECT_TIMEOUT`. Both these parameters are applicable only when the protocol is TCP.

- » The **RETRY\_COUNT** parameter specifies the number of times an address list is traversed before the new connection attempt is terminated. The default value is 0. With respect to SCAN, with **FAILOVER = on**, setting this **RETRY\_COUNT** parameter to a value of 2, for example, means the three SCAN IP addresses are traversed thrice (i.e. 3\*3=9 connect attempts), before the connection is terminated:
  - When the connection request initially comes in, the first randomly assigned IP address tries to service that request, followed by the two remaining IP addresses. (This behavior is controlled by the **FAILOVER** parameter.)
  - The retries then kick in and the list of three IP addresses is tried two more times. **RETRY\_COUNT** is only supported at **DESCRIPTION** level in connect string, but not at global (i.e. `sqlnet.ora`) level.

## Configuring Automatic Failover for ODP.Net Clients

### Prerequisites

Configure Oracle Net alias as described in the section [Configuring OCI Clients for Automatic Failover for OCI Clients](#) above.

### Additional Configuration

1. Enable Fast Connection Failover for ODP.NET connection pools by subscribing to FAN high availability events. To enable Fast Connection Failover, include "HA Events=true" and "pooling=true" in the connection string, as shown in the following example where *user\_name* is the name of the database user and *password* is the password for that user:

```
con.ConnectionString =
    "User Id=user_name;Password=password;Data Source=sales;" +
```

```
"Min Pool Size=10;Connection Lifetime=120;Connection Timeout=60;" +
"HA Events=true;Incr Pool Size=5;Decr Pool Size=2";
```

2. To take advantage of load balancing events with ODP.NET connection pools, set the load balancing attribute in the ConnectionString to TRUE (the default is FALSE). You can do this at connect time. This only works if you are using connection pools, or when the pooling attribute is set to TRUE which is the default.

The following example demonstrates how to configure the ConnectionString to enable load balancing

```
con.ConnectionString =
  "User Id=user_name;Password=password;Data Source=odpapp;" +
  "Min Pool Size=10;Connection Lifetime=120;Connection Timeout=60;" +
  "Load Balancing=true;Incr Pool Size=5;Decr Pool Size=2";
```

3. Configure the service to enable FAN and HA events. This is covered in the [next section](#) of this paper.

## Configuring Oracle Database Services:

On the primary and standby hosts create the service (oltpworkload) that the application will use to connect to the database using the examples provided in this paper. The service should be created such that it is associated with and runs on the database when it is in the 'PRIMARY' database role.

---

*Note: Do not enable TAF on services for JDBC database services that are FAN enabled.*

---

The following example illustrates creating the same services as above for JDBC applications:

» Primary cluster:

```
srvctl add service -db austin -service oltpworkload -preferred ssa1,ssa2,ssa3,ssa4
-role PRIMARY -notifications TRUE -tafpolicy NONE -failovermethod NONE -
failoverdelay 0 -failoverretry 0
```

» Standby cluster:

```
srvctl add service -db houston -service oltpworkload -preferred ssa1,ssa2,ssa3,ssa4
-role PRIMARY -notifications TRUE -tafpolicy NONE -failovermethod NONE -
failoverdelay 0 -failoverretry 0
```

For services that will be used by OCI and ODP.Net applications we modify the above srvctl commands slightly to enable Transparent Application Failover at the service level.

» Primary cluster:

```
srvctl add service -db austin -service oltpworkload -preferred ssa1,ssa2,ssa3,ssa4
-role PRIMARY -notifications TRUE -tafpolicy BASIC -failovermethod BASIC -
failoverdelay 3 -failoverretry 90
```

» Standby cluster:

```
srvctl add service -db houston -service oltpworkload -preferred ssa1,ssa2,ssa3,ssa4
-role PRIMARY -notifications TRUE -tafpolicy BASIC -failovermethod BASIC -
failoverdelay 3 -failoverretry 90
```



## Role Transition Timings

The speed at which an Oracle Data Guard role transition can complete has been reduced with every new release of the Oracle Database. This has been accomplished through enhancements that eliminate steps in the process that consume extra time and that defer non-essential tasks to post-failover, after the new primary database is open and able to process new transactions.

The streamlining of role transition processing is evident by the use of a single DDL to execute switchover and failovers in Oracle Database 12c. For example:

- » For planned events: SQL> ALTER DATABASE SWITCHOVER TO <target db>
- » For unplanned events: SLQ> ALTER DATABASE FAILOVER TO <target db>

There are also improvements in Oracle Database 12c to quickly verify that a Data Guard standby is in a state where a switchover will succeed in advance of actually performing the switchover.

- » To verify from SQL\*Plus using the verify option in the new switchover statement:

```
SQL> ALTER DATABASE SWITCHOVER TO <target db> VERIFY;
```

- » More comprehensive validation is performed by Data Guard broker to evaluate an expanded set of conditions beyond what can be validated using SQL\*Plus. The Data Guard broker command is:

```
DGMGRL> VALIDATE DATABASE [VERBOSE] database-name;22
```

MAA best practice always recommends using the Data Guard broker<sup>23</sup> for management of a Data Guard configuration. The Data Guard broker is utilized either via its command line interface, DGMGRL, or by using Enterprise Manager. Failover operations can be performed manually or can be done automatically using Fast-Start Failover – capabilities included with the Data Guard broker.

### Database Switchover Timings

MAA tests validated switchover timings for both single instance (non-RAC) and Oracle RAC databases with various numbers of active database connections. Tests also measured the additional time required when there were read-only connections on an active standby database. The results shown in the Figure 1 identify the elapsed time during each phase of switchover processing.

In the most complicated test case of 1000 connections on both and Oracle RAC primary (open read-write) and active standby (open read-only), it took a total of approximately 45 seconds between the time when the original primary database was closed and the new primary was open for new connections.

Switchover completed in approximately 12 seconds in the simpler test case of a single instance standby in mounted mode (no Active Data Guard),

<sup>22</sup> <http://docs.oracle.com/database/121/DGBKR/dgmgrl.htm#DGBKR610>

<sup>23</sup> <http://docs.oracle.com/database/121/DGBKR/toc.htm>

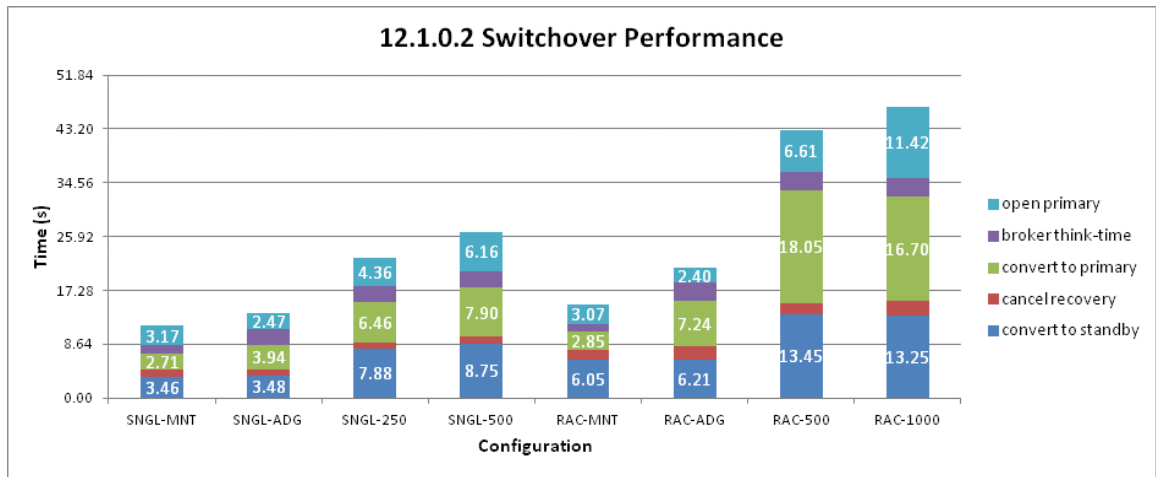


Figure 1: Data Guard Switchover Performance – Details of Timings for Each Phase of Switchover Processing

**Database Failover Timings**

A similar series of tests were run for failover. Failover completes more quickly than a switchover because a sudden outage results in less processing required to transition a standby to the primary role. The results shown in the Figure 2 identify the elapsed time for each phase of failover processing.

In the same complex test case of 1000 connections on both and Oracle RAC primary (open read-write) and active standby (open read-only), it took a total of approximately 26 seconds between when the original primary database outage was detected and the new primary was open for new connections.

Failover completed in approximately 8 seconds in the simpler test case of a single instance standby in mounted mode (no Active Data Guard),

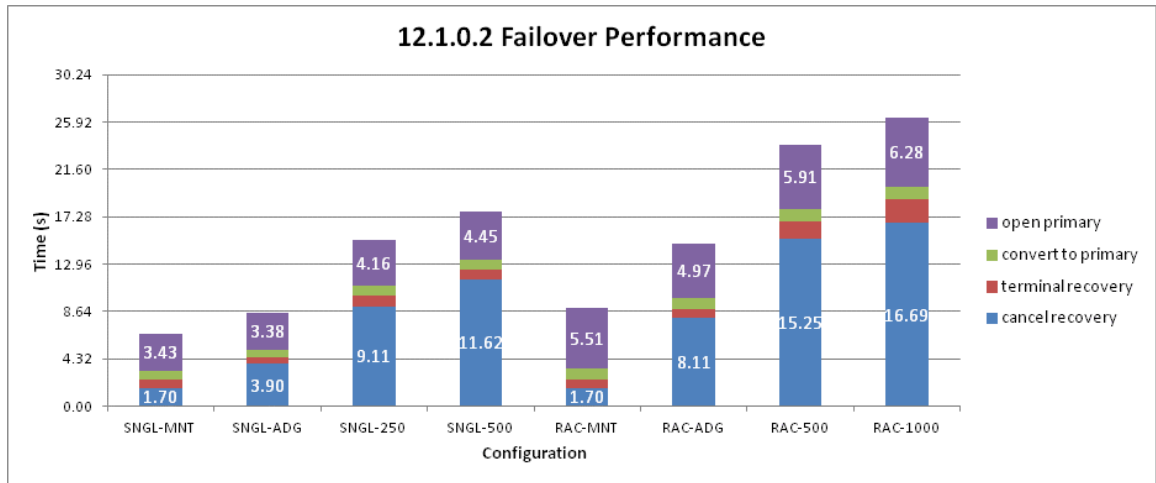


Figure 2: Data Guard Failover Performance – Details of Timings for Each Phase of Switchover Processing

## End-to-End Application Failover Timing with Application Continuity

A final series of tests were performed to measure the time required to transition application connections during a Data Guard failover using Application Continuity. In this test case application connections do not receive an error and in-flight transactions are automatically replayed at the new primary database.

Figure 3 shows what happens to application throughput and session count during tests of a Data Guard failover for an Oracle RAC database with 1000 active connections (there were no read-only connections at the standby database).

This test scenario used the following sequence of events:

1. A shutdown abort was used to simulate an outage of the primary RAC database. Figure 3 shows how quickly database throughput (transactions per second/TPS) and the number of active connections drop to zero.
2. Zero data loss failover to convert the standby database to the primary database role open read-write completes in 24 seconds.
3. The new primary automatically starts all database services appropriate for the primary role.
4. At the same time it publishes a FAN event to notify clients to drop their connections to the failed primary and reconnect. All 1,000 connections complete this process in 5 seconds with throughput quickly restored to pre-outage levels.
5. There is a total elapsed time of 29 seconds between the time when the outage occurs and all 1,000 connections are active at the new primary (the sum of steps 2 through 4).
  - » Note that when Data Guard automatic failover is used the total elapsed time will be lengthened by `FastStartFailoverThreshold` seconds. The lowest recommended threshold value is 6 seconds.
6. The Application Continuity replay driver automatically submits all transactions that were in-flight at the time the outage occurred. The outage manifests to the user as a delayed response with the transaction succeeding without generating an error or requiring the user to resubmit.

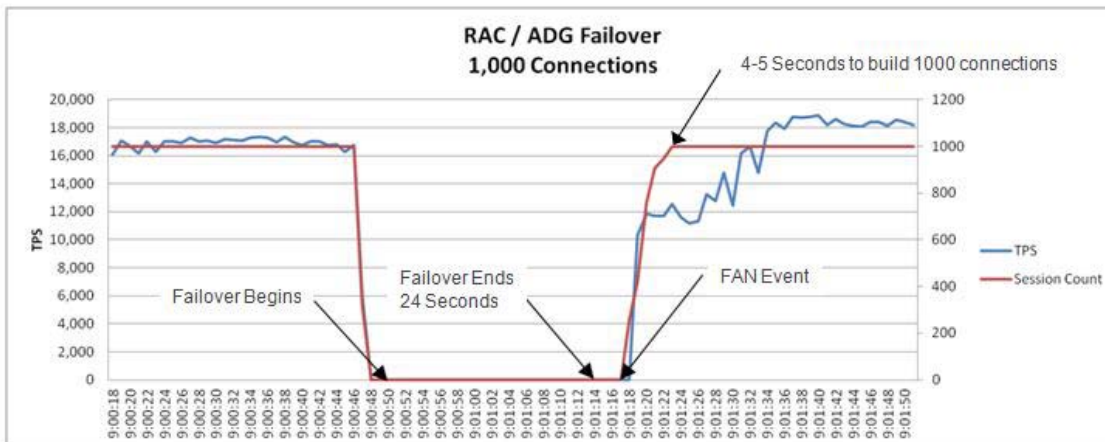


Figure 3: End-to-End Application Failover Timing with Application Continuity

## Automatic Failover for Clients that do not Support FAN

The following sections describe how to configure automatic failover for applications that do not support FAN events. This includes a number of applications from Oracle (e.g. Siebel and Oracle E-Business Suite), and application servers that have not yet implemented FAN support. Even though FAN events cannot be used in such cases, you can still configure applications for efficient failover by using timeouts and application retries.

### OCI Clients that do not Support FAN

For OCI applications, follow the steps described in the section titled [Configuring OCI Clients for Automatic Failover](#), with the exception of the steps related to enabling FAN support (steps 1 through 3). Given that your client does not support FAN, you will use TCP timeouts at either the OS level or the Oracle Net level to break client connections out of a TCP hang.

1. Configure your operating system for efficient TCP timeouts on the hosts that run the application layer. The OS TCP timeouts should be set to the amount of time it takes for the database layer to failover and the database services to be started. Consult your operating system manuals for instructions about properly configuring TCP timeout.
2. Application retries are automated by configuring server-side TAF as described in the above section titled [Configuring OCI Database Services](#). If the application cannot use TAF then the application must be configured with reconnection logic in the event of an exception. For example, when a session from the connection pool receives any exception which results in a disconnect (such as an ORA-3113 error) the application should automatically attempt to reconnect that session. The reconnection attempts should be configured such that they will continue for the length of time that it takes to failover the database layer and bring the application services online.
3. It is important to note that TAF can only automate retries for an existing session. To automate retries when a new connection attempt receives an error, use the `RETRY_COUNT` parameter that is part of an Oracle Net `DESCRIPTION_LIST`. This parameter specifies the number of times an `ADDRESS` list is traversed before the connection attempt is terminated.

### JDBC Clients that do not Support FAN

1. Configure your operating system for efficient TCP timeouts on the hosts that run the application layer. The OS TCP timeouts should be set to the amount of time it takes for the database layer to failover and the database services to be started. Consult your operating system manuals for how to properly configure TCP timeout.
2. Configure reconnection logic within the application to respond appropriately in the event of an exception. For example, when a session from the connection pool receives an exception that results in disconnect (such as an ORA-3113 error), the application should automatically attempt to reconnect that session. The reconnection attempts should be configured such that they will continue for the length of time that it takes to failover the database layer and bring the application services online.

## Additional Considerations for Active Data Guard Connections

Oracle Active Data Guard extends basic Data Guard functionality by offering a number of advanced capabilities. One capability of Active Data Guard is enabling read-only access to a physical standby database while the standby continuously applies changes received from the primary database. Thus, a physical standby database can provide disaster protection while it also increases performance, scalability, and return on investment by offloading ad-hoc queries, Web-based access, reporting, and backups from the primary database. This creates additional considerations should a failover occur. For example, a decision must be made if there is enough capacity to support both read-only workload that was running while it was in standby role and the additional work load will inherited should a failover occur and it becomes the primary production database.

## Initial Connections

Read-only queries and reporting applications can be directed to an Active Data Guard standby by creating a database service description that will start a read-only service anytime a database functions in the standby role. Because role-based services are controlled by the Data Guard Broker and CRS (Oracle Restart in the case of single instance databases), services will also be started/stopped appropriate for the database role whenever a Data Guard role transition occurs.

All applications that connect to a Data Guard configuration should connect using an Oracle Net alias that contains SCAN names for both the primary and standby databases in a DESCRIPTION\_LIST. The Oracle Net alias used by read-write applications that connect to the database in the primary role should list the primary SCAN name in the first DESCRIPTION while the Oracle Net alias used by read-only applications that connect to the database in the standby role should have the standby SCAN name listed in the first DESCRIPTION.

## Managing Existing Connections for Unplanned Outages

During steady state, the primary application will have connections to the primary database and the read-only reporting application will have connections to the active standby database. When a role transition occurs, reporting users connected to an active standby database will be disconnected as part of the failover process. Once the standby is transitioned to the primary database role, the primary database service will be started and users will automatically begin to connect to the new primary databases using the process described earlier in this paper.

Failover of the reporting application connections and services running on the standby database is performed manually. Following a failover, you first determine which database you wish to direct the reporting application. Your choices are as follows:

- » If the new primary database has enough capacity to support both the primary application connections and the reporting application connections, then:
  - » Manually start the reporting database service on the primary database.
  - » Once the service is available, restart the reporting application to get connections established.
- » If the primary database does not have enough capacity to support both the primary application and the reporting application, then start the reporting database service on other Active Data Guard standby databases.
  - » If you do not have multiple Active Data Guard standby databases in your Data Guard configuration, then reinstate the old primary database as a new synchronized standby database<sup>24</sup>. This is a fast and simple process if Flashback Database was enabled prior to the failover and the cause of the outage does not prevent its use for reinstating the failed primary. If Data Guard Fast-Start Failover is used, the observer process automatically reinstates the former primary as a standby database once the problem has been repaired and the database mounted<sup>25</sup>. Once the new standby is operational, start the read-only database service and connect your reporting application.

---

*Note: Oracle Global Data Services (GDS) with Oracle Database 12c provides additional automation for relocating read-only services, eliminating the steps to manually restart read-only services described in this section. GDS is outside the scope of this paper. Additional information on GDS can be found on the Oracle Technology Network<sup>26</sup>.*

---

<sup>24</sup> <http://docs.oracle.com/database/121/DGBKR/cli.htm#DGBKR3431>

<sup>25</sup> <http://docs.oracle.com/database/121/DGBKR/sofo.htm#DGBKR395>

<sup>26</sup> <http://www.oracle.com/technetwork/database/features/availability/global-data-services-1949717.html>

## Controlling Logon Storms

Small connection pools are strongly recommended but when you have many connections controlling logon storms can be done via tuning. The tuning discussed below, done on a single Exadata compute node, shows that when using default TCP queuesize with 5000 simultaneous connections, 80% clients fail to connect and the rest get connected in 46 seconds. After simple TCP queuesize tuning there are zero client-side errors and all clients get connected in 6-8 seconds.

- » Increase the Listen backlog at the OS level. To have the new value take affect without rebooting the server perform he following as root:

```
echo 8000 > /proc/sys/net/core/somaxconn
```

In order to persist the value across reboots, also add to /etc/sysctl.conf:

```
net.core.somaxconn=6000
```

- » Increase Queuesize for listener. Update sqlnet.ora in Oracle home that the listeners are running from to increase the queuesize parameter:

```
TCP.QUEUESIZE=6000
```

Note that this is a new global parameter, in sqlnet.ora, and in case of RAC, needs to be set in the relevant GI/DB home from where listener is running.


There was also a previous parameter (QUEUESIZE=) in each listener address. This works as well and can be used for non-CRS managed listeners. With Oracle RAC 11.2 and later releases, however, CRS Agent manages the listener addresses dynamically, and queuesize is not supported by srvctl. This is why TCP.QUEUESIZE was added in 11.2.0.4, and it is available via a one-off patch for earlier releases. If both parameters are set, then the value in listener address takes precedence.

Rate\_limit is the converse, and can be used in conjunction with the tuning discussed above. If the system can support a large number of concurrent connects with tuning, there is no need to enable rate limit, otherwise rate\_limit can be set to an appropriate value as per the machine capability. A scenario where rate\_limit is useful (and one for which it was originally designed) is when there are database bottlenecks which lead to secondary errors and adverse effects on the server.

## Conclusion

The Oracle Maximum Availability Architecture (MAA) is the best practices blueprint for implementing Oracle high availability technologies. The core principle of high availability is to make outages as transparent as possible to the user. Oracle MAA with Oracle Database 12c is unique in enabling users to continue processing transactions (applications do not receive an error and in-flight transactions succeed) through component, system, and database outages. Oracle Real Application Cluster (Oracle RAC), Oracle Data Guard, Oracle Active Data Guard, and Application Continuity are key components of Oracle MAA that enable this capability.

This technical white paper described Oracle Database 12c configuration best practices to automatically transition application connections from a failed primary database to a new primary database after a Data Guard / Active Data Guard role transition has occurred. Application Continuity provides the additional benefit of preserving in-flight





transactions following a RAC node transition or Data Guard failover, improving quality of service by masking the impact of an outage from the user.



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

Worldwide Inquiries  
Phone: +1.650.506.7000  
Fax: +1.650.506.7200

CONNECT WITH US

-  [blogs.oracle.com/oracle](http://blogs.oracle.com/oracle)
-  [facebook.com/oracle](http://facebook.com/oracle)
-  [twitter.com/oracle](http://twitter.com/oracle)
-  [oracle.com](http://oracle.com)

**Hardware and Software, Engineered to Work Together**

Copyright © 2015, Oracle and/or its affiliates. All rights reserved. This document is provided for information purposes only, and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document, and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group. 0815