## Contents

Executive Overview .............................................. 2  
Introduction ..................................................... 3  
Experience before Application Continuity ................. 4  
Oracle Database 12c Experience using Application Continuity 6  
Application Continuity with Oracle Database 12c ............ 7  
  Application Continuity Coverage .......................... 7  
  Processing Phases in Application Continuity ............... 8  
  Restrictions using Application Continuity ................. 9  
  When is a request not covered for replay .................. 10  
  Potential Side-Effects ..................................... 10  
Assessment Steps to Use Application Continuity ............ 11  
  Assessment steps for your application ..................... 11  
  Check request boundaries .................................. 11  
  Remove Oracle JDBC Concrete Classes ..................... 12  
  Decide if any requests should be disabled ................. 12  
  Register callbacks if state is set outside requests ....... 12  
  Grant support for keeping mutable values ................ 12  
Configuring Application Continuity ......................... 13  
  Configure Oracle JDBC 12c Replay Driver ................. 13  
  Configure Connections for High Availability .......... 14  
  Configure Services for Application Continuity .......... 15  
  Check Resource Allocation ................................. 16  
Administration .................................................. 17  
  Planned Outages ........................................... 17  
  Killing or Disconnecting a Session without Replay ....... 17  
Conclusion ....................................................... 17  
Appendix - New Database Concepts for Application Continuity 18
Executive Overview

It is complex for application developers to mask outages of a database session (instance, node, storage or network or any other related component) and as a result errors and timeouts are often exposed to the end users leading to user frustration, lost productivity, and lost opportunities. Application Continuity masks outages from end users and applications by recovering the in-flight work for impacted database sessions following outages. Application Continuity performs this recovery beneath the application so that the outage appears to the application as a slightly delayed execution.

Application Continuity is invoked for outages that result in recoverable errors, typically related to underlying software, foreground, hardware, communications, network, or storage layers. Application Continuity is used to improve the user experience when handling both unplanned outages and planned outages.

Introduced in Oracle Database 12c, Application Continuity strengthens the fault tolerance of systems and applications that use an Oracle database. Application Continuity is available for –

- Oracle WebLogic Server
- Oracle Universal Connection Pool, used standalone or as a data source for a third party Application Server
- Standard 3rd Party JDBC application servers using the JDBC PooledConnection interface
- Oracle JDBC-Thin Driver

“What application continuity brings to applications now is that they can run in a clustered environment with the security knowing that the application continuity capabilities in Oracle Database 12c are going to handle a lot of failure scenarios automatically.”

— Marc Fielding, ATCG Principal Consultant Oracle, Pythian
Introduction

Application Continuity enables replay, in a non-disruptive and rapid manner, of a database request when a recoverable error makes the database session unavailable. The request can contain transactional and non-transactional calls to the database and calls that are executed locally at the client or middle tier. After a successful replay, the application can continue where that database session left off. Users are no longer left in doubt, not knowing what happened to their funds transfers, flight bookings, and so on; administrators no longer need to re-boot mid-tier machines and recover from logon storms caused by failed sessions. With Application Continuity, the end user experience is improved by masking many outages, planned and unplanned, without requiring the application developer to attempt to recover the request.

Without Application Continuity, it can be almost impossible for an application to mask outages in a safe way, for reasons that include the following:

• The state at the client remains at present time, with entered data, returned data, and variables cached while the state changes reflected in the database session are lost.
• If a commit has been issued, the commit message is not durable. Furthermore, checking the status of a lost request is no guarantee that it will not commit in the future.
• Non-transactional database session states that the application needs to operate are lost.
• If the request can continue, the database and the database session must be in sync.

Application Continuity, however, does this work for the application developer. Application Continuity improves developer productivity by attempting to mask database-related outages that can be safely masked.

Note that applications must continue to include error handling for these cases:

• Non-recoverable errors, such as invalid input data. (Application Continuity applies to recoverable errors.)
• Recoverable errors when replay encounters one of the listed restrictions, such as usage of oracle.sql concrete classes in the application, or when replay is not able to restore the original user-visible results on which the application may have made decisions so far.
Experience before Application Continuity

Without Application Continuity, database recovery does not mask outages that are caused by network outages, instance failures, hardware failures, repairs, configuration changes, patches, and so on.

Earlier Experience for Unplanned Outages

Figure 1 illustrates the earlier experience where errors can be returned to the end user, even when the request completed.

Database outages can cause in-flight work to be lost, leaving users and applications in doubt:
- Users receiving errors
- Users relogging in and reentering data
- Duplicate submission of requests
- Rebooting of mid-tier machines

For end users database session outages can affect them by:

Confusion: Users do not know what happened to their application’s requests like funds transfers, orders, payments, bookings and so on.

Decreased Usability: Users may see an error, lose uncommitted data, and need to log in again and re-enter or resubmit their data.

Disruption: DBAs sometimes need to reboot mid-tier machines to handle the load balance and incoming logon storm.
Earlier Experience for Planned Outages

**Figure 2. Earlier experience for planned outages**

Planned outages are more frequent than unplanned outages.

For applications using RAC, RAC One, Oracle Restart, and Oracle Data Guard with FAN in conjunction with Oracle connection pools – Oracle WebLogic Server Connection Pool, Oracle Universal Connection Pool, Oracle JDBC Implicit Connection Cache (ICC), ODP.NET, OCI Session Pool – planned outages have been masked since Oracle Database 10g Release 2. If you have a third party application server, consider using the Oracle Universal Connection Pool as the data source, for example with IBM WebSphere or Apache Tomcat.

Configure Fast Connection Failover (FCF) at the client. Fast Application Notification (FAN) is pre-configured at the server at install and upgrade for Real Application Clusters.

Use SRVCTL or GDSCTL to relocate the service from an instance or, if you are using a UNIFORM service, then stop the service on an instance. Do not use the -force flag with any of these commands.

FAN sends a notification that the service is down for planned reasons.

On receipt of the FAN event by the connection pool, idle connections to the down service are removed, and no further connections to that instance are allowed. For those applications that return their connections to the pool, the connection pool automatically terminates the connection at the database request boundary. The application and users see no errors. Existing connections on other instances remain usable, and new connections are opened to these instances as needed.

All applications should return their connections to the pool, but in reality this does not always happen. Those that do not can encounter errors when the instance is stopped, leading to a poor user experience.
Oracle Database 12c Experience using Application Continuity

Figure 3 illustrates the improved user experience possible with Oracle Database 12c for applications using Application Continuity.

- Replays in-flight work on recoverable errors
- Masks many hardware, software, network, storage errors and outages
- Improves end user experience and developer productivity

When replay is successful, Application Continuity masks many recoverable database outages from the applications and the users. It achieves the masking by restoring the database session, the full session (including session states, cursors, variables), and the last in-flight transaction (if there is one).

If the database session becomes unavailable due to a recoverable error, Application Continuity attempts to rebuild the session and restore any open transactions to the correct states.

If the transaction is successfully committed and does not need to be re-executed, the successful status is returned to the application.

If the replay is successful, the request continues safely, with no risk of duplication.

If the replay is not successful, the database rejects the replay and the application receives the original error. To be successful, the replay must return to the client the exact same data that the client received previously in the request, which the application potentially made a decision on.
Application Continuity with Oracle Database 12c

Application Continuity Coverage

Application Continuity for Oracle Database 12c supports the following client and server features:

Oracle Database 12c Client

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.

Oracle Universal Connection Pool (UCP), 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.

If using a third party, Java-based application server, there are two ways to obtain Application Continuity functionality out of the box:

• The most effective method is to replace the data source with UCP. This approach is supported by many application servers including IBM WebSphere, Apache Tomcat, and Red Hat JBoss. Using UCP as the data source allows UCP features such as Fast Connection Failover, Runtime Load Balancing and Application Continuity to be used.

• Starting with Oracle JDBC 12102, the request boundaries required for Application Continuity are embedded in the JDBC PooledConnection interface. This makes Application Continuity available to third-party, Java connection pools or standalone Java applications that use this interface in their native pools including IBM WebSphere and Apache Tomcat.

Oracle Database 12c Server

Database request identification is demarcated at the database using APIs beginRequest and endRequest. Identifying requests is transparent when using Oracle connection pools and 3rd party connection pools using the JDBC PooledConnection interface (12.1.0.2 and later).

Call types - SELECT, PL/SQL, ALTER SESSION, DML, DDL, COMMIT, ROLLBACK, SAVEPOINT, JDBC RPCs and local JDBC calls

Transaction types - local, parallel, remote, distributed, and transactions embedded within PL/SQL

Mutable Oracle functions (see Appendix - New Database Concepts for Application Continuity)

Hardware acceleration on hardware using Intel & Sparc chips
Processing Phases in Application Continuity

There are three distinct processing phases used by Application Continuity, shown in table 1. This processing occurs beneath the application in the replay driver and database, and is transparent to the application.

<table>
<thead>
<tr>
<th>NORMAL RUNTIME</th>
<th>RECONNECT</th>
<th>REPLAY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifies database requests</td>
<td>Ensures request has replay enabled</td>
<td>Replays held calls</td>
</tr>
<tr>
<td>Decides what is replayable and what is not</td>
<td>Handles timeouts</td>
<td>During replay, ensures that user visible results match original</td>
</tr>
<tr>
<td>Builds proxy objects</td>
<td>Creates a new connection</td>
<td>Continues the request if replay is successful</td>
</tr>
<tr>
<td>Holds original calls with validation</td>
<td>Validates target database</td>
<td>Throws the original exception if replay is unsuccessful</td>
</tr>
<tr>
<td>Uses Transaction Guard to enforce last outcome</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Normal Runtime - At normal runtime, each database request is tagged with a request begin and end, either by checking out of or back into the WebLogic Server Connection Pool or Universal Connection Pool, or by adding begin and end request tags to your own application or to your own Java connection pool at connection check-out and check-in.

In collaboration between the Oracle JDBC replay driver and the Oracle Database 12c, it is decided which calls in a request are replayable. Replayable calls are held longer by the replay driver, together with validation received from the database. The replay driver holds these calls and validation information until the end of the database request, or until replay is disabled. Replay can be disabled by a restricted call, a commit (in the default mode), the request ending, or explicitly by the application.

Reconnect - The reconnect phase of Application Continuity is triggered by the replay driver when a recoverable error occurs. In this phase, the request is checked to see if replay is still enabled, and the timeout after which replay is not permitted (replay initiation timeout) is checked to ensure that it has not expired. Assuming these checks pass, a new connection to the database is established. The reconnection to the database can take time if the service needs to be re-established, so the DBA should check FAILURE_DELAY and FAILURE_TIMEOUT attributes of the service.

After the replay driver has established a connection to the database, the replay driver checks if the connection is to a valid database target, and whether or not the last transaction (if there was one) committed successfully. Replay will not occur if the connection is to a logically different database or is to the same database but that database has lost transactions. For example, the database has been restored back in time. The replay driver will not resubmit committed transactions. At-most-once submission is enforced using Transaction Guard.

Replay - The replay phase starts once a new connection to the database is established. All calls held by the replay driver are replayed. Replay is disabled if there are any user visible changes in results observed during the replay as determined by the validation. Replay does not allow commit during the replay phase but does allow the last call to commit. That is, the call that encountered the recoverable error. Following a successful replay, the request continues from the point where it had failed.
Restrictions using Application Continuity

The following restrictions apply to using Application Continuity at three levels: global, local, and database.

### TABLE 2. THREE LEVELS OF RESTRICTIONS APPLY FOR APPLICATION CONTINUITY

<table>
<thead>
<tr>
<th>GLOBAL</th>
<th>REQUEST</th>
<th>TARGET DATABASE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Does not support:</td>
<td>For Java streams, replay is on a “best effort” basis</td>
<td>Does not support:</td>
</tr>
<tr>
<td>- XA</td>
<td>- Request-level disable for Active Data Guard with read/write database links</td>
<td>- Logical Standby</td>
</tr>
<tr>
<td>- JDBC concrete classes</td>
<td>- Request-level disable for</td>
<td></td>
</tr>
<tr>
<td>- Default database service</td>
<td>- Alter System</td>
<td></td>
</tr>
<tr>
<td>- 3rd party statement cache</td>
<td>- Alter Database</td>
<td></td>
</tr>
<tr>
<td>- For Java streams, replay is on a “best effort” basis</td>
<td></td>
<td>- Golden Gate</td>
</tr>
<tr>
<td>- Request-level disable for Active Data Guard with read/write database links</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Request-level disable for</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Alter System</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Alter Database</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Global** - This restriction prevents Application Continuity from being enabled or used on any request.

Replay is not supported for applications developed using XA. For example, a Java EE application deployed to WebLogic Server with XA enabled.

For applications using Oracle JDBC Driver, there is no support for oracle.sql concrete classes like BLOB, CLOB, BFILE, OPAQUE, ARRAY, STRUCT or ORADATA. (See MOS note 1364193.1)

Replay is not supported for connections using the database service, i.e. the default service corresponding to the DB_NAME or DB_UNIQUE_NAME. The database service is not intended for use by high availability applications because this service cannot be enabled, disabled, or failed over.

If a statement cache at the application server level is enabled (for example, the WebLogic or third-party application server statement cache), this must be disabled when the replay is used. Instead, configure the JDBC statement cache which supports Application Continuity, and performs better because it is optimized for JDBC and Oracle. Use oracle.jdbc.implicitstatementcachesize=nnn.

**Request** - This restriction disables Application Continuity for part of a database request.

For JDBC stream arguments, replay is on a “best effort” basis. For example, if the application is using physical addresses, the address is no longer valid with the outage and cannot be repositioned.

Replay is not supported if you are using Active Data Guard with read/write database links to another database.

Replay is disabled if the request executes an ALTER SYSTEM or ALTER DATABASE statement. For example, replay will not add table spaces or startup or shutdown the database. Replay is also disabled for some ALTER SESSION statements such as those that change commit behaviour or set events.

**Target Database** - Application Continuity does not support failover to logically different database — including Oracle Logical Standby and Oracle Golden Gate. Replay has a strict requirement that it applies to databases with verified no transaction loss.
When is a request not covered for replay

The following events can disable replay for a database request. If disabled, replay is implicitly re-enabled at the beginning of the next request. Disabling replay does not impact normal runtime or other requests. Validation and recording simply stop and held calls are released by the replay driver. However, that request is not protected for replay once disabled.

### TABLE 3. WHEN IS AN APPLICATION NOT COVERED BY REPLAY

<table>
<thead>
<tr>
<th>NORMAL RUNTIME</th>
<th>RECONNECT</th>
<th>REPLAY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any calls in same request after –</td>
<td>• Error is not recoverable</td>
<td></td>
</tr>
<tr>
<td>• successful commit in dynamic mode (the default)</td>
<td>• Timeouts</td>
<td></td>
</tr>
<tr>
<td>• a restricted call</td>
<td>— Replay initiation timeout</td>
<td></td>
</tr>
<tr>
<td>• disableReplay API</td>
<td>— Max connection retries</td>
<td></td>
</tr>
<tr>
<td></td>
<td>— Max retries per incident</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Target database is not valid for replay</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Last call committed in dynamic mode</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Validation detects different results</td>
<td></td>
</tr>
</tbody>
</table>

**Normal Runtime** - Call capture is enabled at the `beginRequest` tag. Capture happens on all original calls until a successful COMMIT (in dynamic mode), unless a restricted call is encountered, or the application makes an explicit `disableReplay` call. Capture otherwise continues until the `endRequest` tag is encountered.

**Reconnect** - Replay does not occur if the error is not recoverable, if timeouts are exceeded (see Configure Services for Application Continuity), if the target database is not the same or an ancestor of the original database, or if the request committed and Application Continuity is using dynamic mode.

**Replay** - is aborted and the original error is returned if the validation for the replayed call does not pass. The replay must return the same user visible results that the application has seen and potentially made a decision on.

**Potential Side-Effects**

Autonomous transactions and external PL/SQL and server-side Java callouts can cause side effects that are separate from the main database request. Some of the database calls that create side effects include autonomous transactions, email or other notifications using DBMS_ALERT calls, copying files using DBMS_PIPE and RPC calls, writing text files using UTL_FILE calls, making HTTP callouts using UTL_HTTP calls, sending email using UTL_MAIL calls, sending SMTP messages using UTL_SMTP calls, sending TCP messages using UTL_TCP calls and accessing URLs using UTL_URL calls.

Calls with side-effects can leave persistent results behind. For example, if a user walks away part way through some work without committing and the session times out or the user issues Ctrl+C, the foreground or a component fails; the main transaction rolls back while the side effects may have been applied.

Side effects are replayed unless the application specifies otherwise. Applications that use external actions must be reviewed to decide if requests with side effects should be replayed or not. If a request has an external action that should not be replayed, replay should be disabled for that request using the `disableReplay` API. All other requests will continue to be replayed.
Assessment Steps to Use Application Continuity

Assessment steps for your application

Before enabling Application Continuity for an application, complete the assessment steps shown in table 4.

<table>
<thead>
<tr>
<th>DECIDE</th>
<th>WHAT TO DO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request Boundaries</td>
<td>If not using Oracle or 3rd party connection pools that have request boundaries, use UCP as the data source, or add request boundaries</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>JDBC Concrete Classes</td>
<td>Replace JDBC concrete classes with Java interfaces (MOS 1364193.1)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Decide to Disable</td>
<td>Use disableReplay API if a request has any call that should not be replayed</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Callbacks</td>
<td>Ensure that a callback is registered if the state changes outside a request</td>
</tr>
<tr>
<td></td>
<td>Do nothing if using WebLogic Server Active GridLink or UCP labeling.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Mutable Functions</td>
<td>Grant keeping original mutable values if these are compatible with the application</td>
</tr>
</tbody>
</table>

Check Request Boundaries

Request boundaries are tags that mark the beginning and end of a database request. As of Oracle Client 12.1.0.2, pools that embed boundaries include Oracle Universal Connection Pool, all WebLogic Server data sources, and standard 3rd party Application Servers and standalone JDBC pools that use the Oracle 12.1.0.2 JDBC drivers' PooledConnection interface including IBM WebSphere and Apache TomCat.

Stepwise - decide if request boundaries are in place.

1. First determine whether the application borrows and returns connections from one of the pools listed above for each request. If it does, and is returning the connection between requests, no change is needed.

2. If the application is using one of these pools, but is not releasing connections, there is often an application property to set to release the connections. Releasing connections scales much better than not releasing connections, and marks the request boundaries with no other change. Make this configuration change and no other change is needed.

3. If the application uses a third party JDBC connection pool that does not support the JDBC PooledConnection interface, there are two choices:

   - Replace the data source with UCP. See the white papers for this data source replacement on OTN for the various 3rd party application servers under Oracle JDBC. Using this method, allows the 3rd party application server to use all UCP features including FAN, Runtime Load Balancing and Application Continuity. This method is a highly recommended solution for IBM WebSphere and Apache TomCat.

   - Add beginRequest and endRequest APIs to identify database request boundaries. These API’s have no performance cost and are isolated to check-out and check-in for a custom JDBC pool.
Remove Oracle JDBC Concrete Classes

**Determine whether the application uses Oracle JDBC concrete classes.** To use Application Continuity, Oracle JDBC concrete classes must be replaced. For information about the deprecation of concrete classes, including actions to take if an application uses them, see My Oracle Support Note 1364193.1
https://support.oracle.com/CSP/main/article?cmd=show&type=NOT&id=1364193.1

Decide if any requests should be disabled

Application developers should make a conscious decision to enable replay. This is especially important for requests that use the UTL_HTTP package or autonomous procedures, that enforce serialization, or that rely on mid-tier wall clock time. Use the disable replay API (disableReplay) for any request that should not be replayed.

If a request makes an external call using one of the external PL/SQL messaging actions or via server-side JDBC, or uses autonomous transactions, check this request. Decide if it should be replayed or not, and disable if it should not be replayed. See Potential Side Effects.

Replay recovers sessions concurrently and independently. Disable replay if the application logic assumes that otherwise independent database sessions are synchronized. For example, do not use replay if the application synchronizes sessions using volatile resources such as user locks or external devices that are held until commit, rollback, or session loss.

Disable replay if the request relies on time at the mid-tier in the execution logic. If a request assumes that a statement executed at Time T1 is not re-executed at Time T2 then disable replay for this request. The replay driver does not repeat the mid-tier logic. It repeats the database calls that execute as part of this logic.

Register callbacks if state is set outside requests

**Assess if the application sets state outside a database request.** If database session state is set when a user starts a request, outside that request, replay needs to know about it in order to re-execute the calls. Choose one of the following callback options:

- Universal Connection Pool or WebLogic Server Connection Labeling
- Connection Initialization Callback

When using Oracle WebLogic Server or the Universal Connection Pool, connection labeling is recommended. If using connection labeling, these labels are used automatically for Application Continuity. No change is needed.

If using an application’s own callback, register this at the WebLogic Administration Console, or at Universal Connection Pool, or JDBC replay driver levels. Application Continuity will re-execute that callback at replay. Use the connection initialization callback only when the application has not implemented connection labeling, and needs state that is not established in the request.

Grant support for keeping mutable values

Decide whether keeping original mutable values is compatible with the application and if so keep original mutable values for replay. Mutable functions are functions that can return a new value each time they are executed. Support for keeping the original results of mutable functions is provided for SYSDATE, SYSTIMESTAMP, SYS_GUID, and sequence.NEXTVAL. If the original values are not kept and if different values are returned to the application at replay, replay will be rejected.
If the application can use original values, configure mutable objects using GRANT KEEP for application users, and the KEEP clause for a sequence owner. When KEEP privilege is granted, replay applies the original function result at replay.

For example

```sql
GRANT [KEEP DATE TIME | KEEP SYSGUID].. [to USER]
GRANT KEEP SEQUENCE.. [to USER] on [sequence object];
ALTER SEQUENCE.. [sequence object] [KEEP|NOKEEP];
```

Note:

Mutable application applies to the local database. It does not traverse database links. It also does not apply for SYS_GUID if pushed down to parallel query slaves.

**Configuring Application Continuity**

After verifying that the application is ready for replay, configure Application Continuity at the client and at the server, shown in the tasks bar in figure 4.

![Diagram of Configuring Application Continuity](image)

**Figure 4. Configuring Application Continuity**

Configure Oracle JDBC 12c Replay Driver

Choose from one of the following options depending on your configuration:

Configure Oracle Universal Connection Pool 12c

Configure the Oracle JDBC 12c Replay Data Source as a connection factory on UCP PoolDataSource:

```java
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 as shown in the Figure 5 below.

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 -

```java
replay datasource=oracle.jdbc.replay.OracleDataSourceImpl
```

Configure Connections for High Availability

The following points are needed for successful failover and fallback.

The REMOTE_LISTENER setting for the database must include the addresses in the ADDRESS_LISTs for all URL used for client connections:

- 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, RETRY_DELAY (JDBC Oracle client 12102 and later), and CONNECT_TIMEOUT parameters in the URL to allow new incoming connections to retry. This is a general recommendation for configuring the JDBC Thin driver connections for high availability.
url = "jdbc:oracle:thin:@DESCRIPTION_LIST=
(LOAD_BALANCE=off)
(FAILOVER=on)
(DESCRIPTION=(CONNECT_TIMEOUT=90) (RETRY_COUNT=10) (RETRY_DELAY=3)
  (ADDRESS_LIST=
   (LOAD_BALANCE=on)
   (ADDRESS=(PROTOCOL=TCP) (HOST= CLOUD-SCANVIP.example.com)(PORT=1521))))
(DESCRIPTION=(CONNECT_TIMEOUT=90) (RETRY_COUNT=10) (RETRY_DELAY=10)
  (ADDRESS_LIST=
   (LOAD_BALANCE=on)
   (ADDRESS=(PROTOCOL=TCP) (HOST= CLOUD-SCANVIPDG.example.com)(PORT=1521))))
(DESCRIPTION=(CONNECT_TIMEOUT=90) (RETRY_COUNT=10) (RETRY_DELAY=10)
  (ADDRESS_LIST=
   (LOAD_BALANCE=on)
   (ADDRESS=(PROTOCOL=TCP) (HOST= CLOUD-SCANVIPDG.example.com)(PORT=1521))))")

alter system set Remote_listeners="CLOUD-SCANVIP.example.com:1521"

Repeat this to register scan CLOUD-SCANVIPDG.example.com at the data guard site

NOTES – The following notes apply for Java Net Client 12.1.0.2 and earlier

- It is not possible to set TRANSPORT_CONNECT_TIMEOUT. See Bug 19000803.
- Do not use SQLnetDef.TCP_CONN_TIMEOUT as this overrides CONNECT_TIMEOUT.
- RETRY_COUNT for Java Net does not retry on service down. See Bug 19154304

Configure Services for Application Continuity

To use Application Continuity, set the service attributes using one of SRVCTL or GDSCTL or DBMS_SERVICE, depending on your system configuration-

- **FAILOVER_TYPE**: Set this to TRANSACTION to enable Application Continuity
- **COMMIT_OUTCOME**: Set this 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 18 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)
Example

To use SRVCTL to modify the service attributes, use a command similar to the following, where EMEA is the name of the Oracle database, and GOLD is the name of the service:

```
srvcctl modify service -db EMEA -service GOLD -failovertypetransaction
-replay_init_time 300 -failoverdelay 3 -commit_outcome true
```

To use DBMS_SERVICE package, modify service attributes in the following way:

```
declare
    params dbms_service.svc_parameter_array;
begin
    params('FAILOVER_TYPE') := 'TRANSACTION';
    params('REPLAY_INITIATION_TIMEOUT') := 300;
    params('FAILOVER_DELAY') := 3;
    params('FAILOVER RETRIES') := 30;
    params('commit_outcome') := 'true';
    dbms_service.modify_service('[your service]', params);
end;
```

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 reduced on the database side for platforms with current Intel and Sparc chips.
Administration

Planned Outages

For planned outages the recommended approach is to drain requests via FAN for Oracle connection pools, in combination with Application Continuity for those requests that do not complete within the allocated time.

Using RELOCATE or STOP commands with NO force flag, the FAN planned event clears the idle sessions immediately and marks the active sessions to be released at check in (end of request). The FAN event causes most sessions to drain from the instance without disrupting work.

If not all sessions have checked in and the time to stop the instance has been reached, stop the instance (abort). If configured, Application Continuity will attempt to recover those remaining sessions.

Killing or Disconnecting a Session without Replay

When Application Continuity is configured and a DBA kills or disconnects a session, Application Continuity attempts to recover that session. However, if you do not want the session to be replayed, use the NOREPLAY keyword:

**Stopping an individual session with no replay**

```
alter system kill session 'sid, serial#, @inst' noreplay;
alter system disconnect session 'sid, serial#, @inst' noreplay;
execute DBMS_SERVICE.DISCONNECT_SESSION('[service name]',
DBMS_SERVICE.NOREPLAY);
```

**Stopping an individual service with no replay**

With `srvctl stop service` you can specify the instance or the node to stop. Using `-force` and `-noreplay` options will avoid replay if noreplay is required. Examples

```
srvctl stop service -db orcl -instance orcl2 -service orcl_pdb38 -noreplay -force
srvctl stop service -db orcl -node rws3 -service orcl_pdb38 -noreplay -force
```

**Stopping a group of services together with no replay**

Stop all services that can run at a database level, at an instance level or at a node level as a group. Use `-force` and `-noreplay` options if noreplay is required. Examples

```
srvctl stop service -db orcl -noreplay -force
srvctl stop service -db orcl -instance orcl2 -noreplay -force
srvctl stop service -db orcl -node rws3 -noreplay -force
```

Conclusion

Application Continuity attempts to mask outages from applications and end users by replaying incomplete database requests following recoverable outages. Many outages can be masked. This results in fewer calls to the application's error handling logic (i.e. less often that the application raises an error to user, leaves the user not knowing what happened, or forces the user to re-enter data, or worse that administrators must restart mid-tier servers to cope with the failure, etc.). Application Continuity strengthens the fault tolerance of systems and applications that use an Oracle database.
Appendix - New Database Concepts for Application Continuity

The following terms and concepts are used with Application Continuity

**Recoverable Error (enhanced)**

A recoverable error is an error that arises due to an external system failure, independent of the application session logic that is executing. Recoverable errors occur following planned and unplanned outages of foregrounds, networks, nodes, storage, and databases. The application receives an error code that can leave the application not knowing the status of the last operation submitted. Recoverable errors are enhanced in Oracle Database 12c, to include more errors and to include a public API for OCI. Applications should no longer list error numbers in their code. Application Continuity is invoked following a recoverable error code.

**Reliable Commit Outcome**

From the client perspective, a transaction is committed when an Oracle message (termed Commit Outcome), generated after the transaction redo is written, is received by the client. However, the COMMIT message is not durable. Application Continuity uses Oracle Database 12c Transaction Guard to obtain the Commit Outcome reliably when it may have been lost following a recoverable error.

**Database Request**

A database request is a unit of work submitted from the application. It typically corresponds to the SQL and PL/SQL, database RPC calls, and local client-side calls of a single web request on a single database connection. It is generally demarcated by the calls made to check-out and check-in the database connection from a connection pool. For recoverable errors, Application Continuity re-establishes the database session and repeats an uncommitted database request safely.

Typically, database requests that use JDBC follow a standard pattern. Here is a code snippet which shows how many database requests are designed.

1. A database request begins with a `getConnection` call on the `PoolDataSource`.
2. The application’s logic is executed. This could include executing SQL, PL/SQL, RPC, or local calls.
3. The transaction is committed.
4. The database request ends when the connection is returned to the connection pool.
Mutable Functions

Mutable functions are functions that can change their results each time that they are called. Mutable functions can cause replay to be rejected because the results visible to the client can change at replay. Consider sequence.NEXTVAL that is often used in key values. If a primary key is built with a sequence value and this is later used in foreign keys or other binds, the same function result must be returned at replay.

Application Continuity provides mutable value replacement at replay for Oracle function calls if GRANT KEEP or ALTER.. KEEP has been configured. If the call uses database functions that support retaining original mutable values, including sequence.NEXTVAL, SYSDATE, SYSTIMESTAMP, and SYS_GUID, then, the original values returned from the function execution can be saved and reapplied at replay. If an application decides not to grant mutable support and different results are returned to the client at replay, replay for these requests is rejected.
Application Continuity
July 2014
Author: Carol Colrain
Contributing Authors: Tong Zhou, Nancy Ikeda, Stefan Roesch, Jean de Lavarene, Kevin Neel

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

Worldwide Inquiries
Phone: +1.650.506.7000
Fax: +1.650.506.7200

Copyright © 2014, 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.

0814

Hardware and Software, Engineered to Work Together

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