Zero Downtime: Hiding Planned Maintenance and Unplanned Outages from Applications

Carol Colrain
Consulting Member of Technical Staff,
Technical Lead for Client-Failover, RAC Development
Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle’s products remains at the sole discretion of Oracle.
Program Agenda

1. Problems to Solve  10
2. Fast Application Notification  15
3. Continuous Connections  15
4. Hiding Planned Maintenance  15
5. Hiding Unplanned Outages  30
6. Success Stories
What problems confront applications at database outages?
In-Flight Work

Pre-12c Situation

Database outages cause in-flight work to be lost, leaving users and applications in-doubt

- Restart applications and mid-tiers
- User frustration
- Cancelled work
- Duplicate submissions
- Errors even when planned
- Developer pains

Sorry. Internal Server Error - 500 Error
We are currently experiencing an issue with our servers on coolcar.com. Please come back later.
How do we reach all applications?

• Move work to different instance/database with no errors reported to applications at planned maintenance

• Hide unplanned database outages from the applications

• Take adoption out of the developers hands to configuration/operation only

• Work with current drivers and older database, whenever possible
Outage Detection

The dead thing cannot tell you that it’s dead
Applications Waste Time

• Hanging on TCP/IP timeouts
• Connecting when services are down
• Not connecting when services resume
• Receiving errors during planned maintenance
• Processing partial results when server is down
• Attempting work at slow, hung, or dead nodes

Performance issues not reported in your favorite tools.
Fast Application Notification

- **Down** – received in low ms to invoke failover
- **Planned Down** – drains sessions for planned maintenance with no user interruption whatsoever
- **Up** – Re-allocates sessions when services resume
- **Load %** - Advice to balance sessions for RAC locally and GDS globally
- **Affinity** - Advice when to keep conversation locality

**Proven since 10g**

**12c: Auto-Configuration + Global Data Services**
### 12c FAN: Standardized, Auto-Configured

<table>
<thead>
<tr>
<th>Client</th>
<th>10g</th>
<th>11g</th>
<th>12c</th>
</tr>
</thead>
<tbody>
<tr>
<td>JDBC Implicit Connection Cache</td>
<td>ONS</td>
<td>ONS</td>
<td>desupport</td>
</tr>
<tr>
<td>JDBC Universal Connection Pool</td>
<td>ONS</td>
<td></td>
<td>ONS</td>
</tr>
<tr>
<td>OCI/OCCI driver</td>
<td>AQ</td>
<td>AQ</td>
<td>ONS</td>
</tr>
<tr>
<td>ODP.NET Unmanaged Provider (OCI)</td>
<td>AQ</td>
<td>AQ</td>
<td>ONS</td>
</tr>
<tr>
<td>ODP.NET Managed Provider (C#)</td>
<td>ONS</td>
<td></td>
<td>ONS</td>
</tr>
<tr>
<td>OCI Session Pool</td>
<td>AQ</td>
<td>AQ</td>
<td>ONS</td>
</tr>
<tr>
<td>WebLogic Active GridLink</td>
<td>ONS</td>
<td></td>
<td>ONS</td>
</tr>
<tr>
<td>Tuxedo</td>
<td>ONS</td>
<td></td>
<td>ONS</td>
</tr>
<tr>
<td>Listener</td>
<td>ONS</td>
<td>ONS</td>
<td>ONS</td>
</tr>
</tbody>
</table>
12c JDBC FAN Auto-Configures

• **12c JDBC clients and 12c Oracle database**
  – Check ons.jar is included in the class path
  – To enable FAN set the pool property
    • `fastConnection FailoverEnabled=true`

• **Before 12c - JDBC clients or database**
  – also set the pool property for remote ons
    • `oracle.ons.nodes = mysun05:6200,mysun06:6200,mysun07:6200,mysun08:6200`

or via autoons

• `oracle.ons.nodes.001=node1a,node1b,node1c... (site 1 nodes here)`
  `oracle.ons.nodes.002=node2a,node2b,node2c... (site 2 nodes here)`
12c OCI FAN Auto-Configures

• 12c OCI clients and 12c Oracle database
  Use srvctl to configure the service for AQ HA Notification:
  `srvctl modify service -db EM -service GOLD -notification TRUE`
  
  For the client, enable in `oraaccess.xml`

• Before 12c OCI clients or database
  – Enable OCI_EVENTS at environment creation `OCIEnvCreate(..)`
  – Link the app with the client thread o/s library.
FAN with other Java Application Servers

Use UCP – a simple DataSource replacement

IBM WebSphere
Apache Tomcat
See OTN.

Class path to be set for UCP JDBC Provider

$\{(WAS\_INSTALL\_ROOT)}/jdbc/ojdbc7.jar
$\{(WAS\_INSTALL\_ROOT)}/jdbc/ucp.jar
$\{(WAS\_INSTALL\_ROOT)}/jdbc/ons.jar

Pool Data Source
Monitor FAN

- Create a FAN callout in ..$GRID_HOME/racg/userco
- Download FANwatcher from OTN RAC page

FANwatcher

..  
VERSION=1.0 event_type=SERVICEMEMBER service=orcl_swing_pdb2 instance=orcl1 database=orcl db_domain= host=sun01 status=down reason=USER timestamp=2014-07-30 12:02:51 timezone=-07:00
VERSION=1.0 event_type=SERVICEMEMBER service=orcl_swing_pdb10 instance=orcl1 database=orcl db_domain= host=sun01 status=down reason=USER timestamp=2014-07-30 12:02:52 timezone=-07:00
VERSION=1.0 event_type=SERVICE service=orcl_swing_pdb10 database=orcl db_domain= host=sun01 status=down reason=USER
Continuous Connections

Applications should see no errors while services relocate.
Connections Appear Continuous for OCI - 12102 while a service is temporarily unavailable

alias = (DESCRIPTION =
  (CONNECT_TIMEOUT=90) (RETRY_COUNT=20)(RETRY_DELAY=3)
  (TRANSPORT_CONNECT_TIMEOUT=3)
  (ADDRESS_LIST =
    (LOAD_BALANCE=on)
    ( ADDRESS = (PROTOCOL = TCP)(HOST=primary-scan)(PORT=1521)))
  (ADDRESS_LIST =
    (LOAD_BALANCE=on)
    ( ADDRESS = (PROTOCOL = TCP)(HOST=secondary-scan)(PORT=1521)))
  (CONNECT_DATA=(SERVICE_NAME = gold-cloud)))

Safe for failover + storms
Retry while service is unavailable
New

Balance scan
OCI Only

ORACLE
Connections Appear Continuous for Java - 12102

while a service is temporarily unavailable

(DESCRIPTION =

(CONNECT_TIMEOUT= 4)  (**RETRY_COUNT=20**)(**RETRY_DELAY=3**)  
(ADDRESS_LIST =

(LOAD_BALANCE=on) 
 ( ADDRESS = (PROTOCOL = TCP)(HOST=primary-scan)(PORT=1521)))  
(ADDRESS_LIST =

(LOAD_BALANCE=on)  
 ( ADDRESS = (PROTOCOL = TCP)(HOST=secondary-scan)(PORT=1521)))

(CONNECT_DATA=(**SERVICE_NAME** = gold-cloud)))

Lower skips down IPs but can cause timeouts on failover and storms

Balance scan
Connections Appear Continuous **for ODP.NET - 12102**
while a service is temporarily unavailable

**Increase ODP.NET “connection timeout”** connection attribute for failover to complete –
e.g. 90s to accommodate login storms.

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

- **ALWAYS use application services to connect to the database.**
  - Do not use the database service or PDB service – these are for administration only, not HA

- Use current client driver (12102) with current or older RDBMS

- Use one DESCRIPTION – more cause long delays connecting

- Set CONNECT_TIMEOUT=90 or higher to prevent logon storms (OCI and ODP)
  - Do not also set JDBC property oracle.net.ns.SQLnetDef.TCP_CONNTIMEOUT_STR as it overrides

- LOAD_BALANCE=on per address list balances SCANs

- Do not use retry count without retry delay

- **Do not use Easy*Connect – it has no HA capabilities.**
Patches before 12.2

For Java Net Connections only:

• RETRY_COUNT must apply when service is down (19154304)
  – PSE 21439688 on 12.1.3.1 WebLogic Server

• Set LOAD_BALANCE=on per address to balance the SCAN (18057904)

• 11 Databases - NO DELAY PARAMETER FOR RETRYING INCOMING CONNECTIONS (16618074)

• TRANSPORT_CONNECT_TIMEOUT (19000803)
Transparent Planned Maintenance

Applications should see no errors during maintenance.
Transparent Planned Maintenance

• Drains work away from instances targeted for maintenance initiated by FAN
  – Supports well behaved applications using Oracle pools
    ▪ WebLogic Active GridLink, UCP, ODP.NET unmanaged and managed, OCI Session Pool, PHP
    ▪ 3rd party application servers using UCP DataSource: IBM Websphere, Apache Tomcat,…

• Failover at transactional disconnect
  ▪ applications adapted for TAF SELECT with OCI or ODP.Net unmanaged provider
  ▪ applications with own/custom failover
DBA steps - Drain Work at Safe Places

Repeat for each service allowing time to drain

- **Stop service (no –force)**
  
  ```bash
  srvctl stop service -db .. -instance .. [-service] .. (omitting -service stops all)
  ```

- **or Relocate service (no –force)**
  
  ```bash
  srvctl relocate service -db .. -service .. -oldinst .. -newinst
  srvctl relocate service -db .. -service .. -currentnode.. -targetnode
  ```

- **Wait to allow sessions and XA branches to drain. (see notes)**

- **For remaining sessions, stop transactional per service**
  
  ```bash
  exec dbms_service.disconnect_session('[service]', DBMS_SERVICE.POST_TRANSACTION);
  ```

- **Now stop the instances using your preferred method including opatch**

- **For major maintenance operations, disable to prevent restarts**
  
  ```bash
  srvctl disable instance -db .. -instance
  ```
## How it works

| Applications using ...                      | Oracle pools or drivers – WebLogic Active GridLink, UCP, ODP.NET managed/unmanaged, OCI, Tuxedo  
<table>
<thead>
<tr>
<th></th>
<th>3rd party App Servers using UCP: IBM WebSphere, Apache Tomcat</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DBA Step</strong></td>
<td>`srvctl [relocate</td>
</tr>
<tr>
<td><strong>Sessions Drain</strong></td>
<td>Immediately</td>
</tr>
<tr>
<td></td>
<td>New work is redirected by listeners</td>
</tr>
<tr>
<td></td>
<td>Idle sessions are released</td>
</tr>
<tr>
<td></td>
<td>Active sessions are released when returned to pools</td>
</tr>
</tbody>
</table>
Planned Maintenance at NEC

WebLogic Active GridLink and Real Application Clusters

1. `srvctl stop services at one instance & drain (e.g. 5-7s)`
2. Instance shutdown
3. Apply patch or change parameter or other maintenance
4. Restart instance & service

No errors, application continues
Planned Maintenance at NEC

WebLogic Active GridLink and Data Guard

1. `srvctl stop services on primary site & drain (e.g. 25s – 30s)`
2. Data Guard switchover
3. New primary database open, start service, rebalance

No errors, application continues
## High Availability by Patch Type

<table>
<thead>
<tr>
<th></th>
<th>One-Off</th>
<th>PSU/CPU</th>
<th>Bundle Patch</th>
<th>Patch Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAC Rolling</td>
<td>96%</td>
<td>All</td>
<td>Most</td>
<td>No</td>
</tr>
<tr>
<td>Standby First</td>
<td>98%</td>
<td>All</td>
<td>All</td>
<td>No</td>
</tr>
<tr>
<td>Out of Place</td>
<td>All</td>
<td>All</td>
<td>Exadata bundles</td>
<td>No</td>
</tr>
<tr>
<td>Online - Hot</td>
<td>82%*</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

*Available from 11.2.0.2 onward*
## Enterprise Applications

<table>
<thead>
<tr>
<th>Application</th>
<th>DBA operation at planned maintenance</th>
<th>Configuration Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Siebel</td>
<td>disconnect sessions transactional</td>
<td>NET</td>
</tr>
<tr>
<td>PeopleSoft</td>
<td></td>
<td>NET and TAF SELECT</td>
</tr>
<tr>
<td>JD Edwards</td>
<td></td>
<td>NET</td>
</tr>
<tr>
<td>Informatica</td>
<td></td>
<td>NET</td>
</tr>
</tbody>
</table>
Application Continuity

Unplanned outages should be hidden from applications
Application Continuity

In-flight work continues

- Replays in-flight work on recoverable errors
- Masks most hardware, software, network, storage errors and outages
- Supports JDBC-Thin, UCP, WebLogic Server, 3rd Party Java app servers
- RAC, RAC One, & Active Data Guard
- Improves end user experience
Database Request – UCP example

PoolDataSource pds = GetPoolDataSource();
Connection conn = pds.getConnection();
PreparedStatement pstmt = ... 
... 
SQL, PL/SQL, local calls, RPC 
... 
conn.commit(); 
conn.close();

Request Begins

Request Body often ends with COMMIT

Request Ends
Phases in Application Continuity

1 – Normal Operation
• Client marks database requests
• Server decides which calls can & cannot be replayed
• Directed, client holds original calls, their inputs, and validation data

2 – Outage Phase 1: Reconnect
• Checks replay is enabled
• Verifies timeliness
• Creates a new connection
• Checks target database is valid
• Uses Transaction Guard to force last outcome

3 – Outage Phase 2: Replay
• Replays captured calls
• Ensures results returned to application match original
• On success, returns control to the application
Exclusions
When replay is not enabled

**Application Level**
- Default database or default PDB service
- Deprecated, non-standard JDBC classes
- XA in 12.1

**Request Level**
- Admin actions
  - Alter system
  - Alter database
  - Alter session (subset)
- Best effort for streams; OCI only – no ADT’s or AQ
- Active Data Guard with read/write DB links

**Target Database**
- Databases able to diverge
  - Logical Standby
  - Golden Gate
  - PDB Clone
## Steps to use Application Continuity

<table>
<thead>
<tr>
<th>Check</th>
<th>What to do</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request Boundaries</td>
<td>UCP, WebLogic, and supported 3rd Party App servers – return connections to pool</td>
</tr>
<tr>
<td>JDBC Deprecated Classes</td>
<td>Replace non-standard classes (MOS 1364193.1); use assessment to know</td>
</tr>
<tr>
<td>Side Effects</td>
<td>Use disable API if a request has a call that should not be replayed</td>
</tr>
<tr>
<td>Callbacks</td>
<td>Register a callback for applications that change state outside requests For WebLogic Active Gridlink and UCP labels – do nothing</td>
</tr>
<tr>
<td>Mutable Functions</td>
<td>Grant keeping mutable values, e.g. sequence.nextval</td>
</tr>
</tbody>
</table>
Request Boundaries

Let the database know that it has a request

• Oracle Pools – JDBC UCP and WebLogic
  • Return connections to pool

• 3rd Party Java Application Servers
  IBM WebSphere, Apache Tomcat, your own
  • Use UCP – a simple DataSource switch
  • Return connections to pool

• Custom - Standalone Java, 3rd Party
Disabling Replay

Use `disableReplay` API for requests that should not be replayed.

Make a conscious decision to replay side effects
  e.g. Autonomous Transactions
  UTL_HTTP
  UTL_URL
  UTL_FILE
  UTL_FILE_TRANSFER
  UTL_SMTP
  UTL_TCP
  UTL_MAIL
  DBMS_JAVA callouts
  EXTPROC
Grant Mutables

Keep original function results at replay

For owned sequences:

- ALTER SEQUENCE.. [sequence object] [KEEP|NOKEEP];
- CREATE SEQUENCE.. [sequence object] [KEEP|NOKEEP];

Grant and Revoke for other users:

- GRANT [KEEP DATE TIME | KEEP SYSGUID].. [to USER]
- REVOKE [KEEP DATE TIME | KEEP SYSGUID][from USER]
- GRANT KEEP SEQUENCE on [sequence object] [to USER] ;
- REVOKE KEEP SEQUENCE on [sequence object] [from USER]
Callbacks

For applications that set state outside database requests

• WebLogic and UCP Connection Labeling
  – Do nothing

• Custom
  – Register Connection Initialization Callback
  – Sets initial state for a session at BOTH runtime and replay
  – Available with WebLogic, UCP, JDBC-Thin driver
Configuration at Database

Set Service Attributes

FAILOVER_TYPE = TRANSACTION for Application Continuity

Review the service attributes:

- COMMIT_OUTCOME = TRUE for Transaction Guard
- REPLAY_INITIATION_TIMEOUT = 300 after which replay is canceled
- FAILOVER_RETRIES = 30 for the number of connection retries per replay
- FAILOVER_DELAY = 3 for delay in seconds between connection retries
Configuration at Client

Use JDBC Replay Data Source

At WebLogic Console or UCP, or your own property file –

Select new 12.1 datasource
replay datasource=oracle.jdbc.replay.OracleDataSourceImpl

Use JDBC statement cache rather than the WLS Statement Cache
# Killing Sessions - Extended

<table>
<thead>
<tr>
<th>DBA Command</th>
<th>Replays</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>srvctl stop service -db orcl -instance orcl2 -force</code></td>
<td>YES</td>
</tr>
<tr>
<td><code>srvctl stop service -db orcl -node rws3 -force</code></td>
<td>YES</td>
</tr>
<tr>
<td><code>srvctl stop service -db orcl -instance orcl2 -noreplay -force</code></td>
<td></td>
</tr>
<tr>
<td><code>srvctl stop service -db orcl -node rws3 -noreplay -force</code></td>
<td></td>
</tr>
<tr>
<td><code>alter system kill session ... immediate</code></td>
<td>YES</td>
</tr>
<tr>
<td><code>alter system kill session ... noreplay</code></td>
<td></td>
</tr>
<tr>
<td><code>dbms_service.disconnect_session([service], dbms_service. noreplay)</code></td>
<td></td>
</tr>
</tbody>
</table>
Application Continuity Performance
WebLogic Server Active GridLink and Real Application Clusters

Response time (ms)

Throughput (tx/s)

CPU per transaction

Memory per transaction

select & update

0

select & update

0

AC OFF

AC ON

MedRec

Application

AP server CPU

DB servr CPU

AP server memory

select & update

0

select & update

0

600

400

200

0

60

10

20

30

40

50

60

Throughput (tx/s)

Throughput (tx/s)

Response time (ms)

Response time (ms)

CPU per transaction

CPU per transaction

Memory per transaction

Memory per transaction

AP server CPU

DB servr CPU

AP server CPU

DB servr CPU

AP server memory

AP server memory

0

0

600

400

200

0

60

10

20

30

40

50

60

select & update

select & update

Response time (ms)

Response time (ms)

Throughput (tx/s)

Throughput (tx/s)

CPU per transaction

CPU per transaction

Memory per transaction

Memory per transaction

AP server CPU

DB servr CPU

AP server CPU

DB servr CPU

AP server memory

AP server memory
AC Assessment – in ORAchk

How effective is Application Continuity for user application
Where Application Continuity is not in effect - what steps need to be taken
When Application Continuity cannot be used and why due to a global restriction

<table>
<thead>
<tr>
<th>No</th>
<th>Assessment functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Pretest(sanity check)</td>
</tr>
<tr>
<td>1</td>
<td>JDBC Concrete Classes</td>
</tr>
<tr>
<td>2</td>
<td>Request Boundaries and Protection Level</td>
</tr>
<tr>
<td>3</td>
<td>Decide to Disable</td>
</tr>
<tr>
<td>4</td>
<td>Callbacks</td>
</tr>
<tr>
<td>5</td>
<td>Mutable Functions</td>
</tr>
</tbody>
</table>

Available May 2015 ORAchk
AC Statistics

Supported for Oracle JDBC replay driver

Statistics are client-side, cumulative per-connection or total for all pooled connections using oracle.jdbc.replay.ReplayableConnection

ReplayableConnection.getReplayStatistics (FOR_CURRENT_CONNECTION) returns statistics for current connection

ReplayableConnection.getReplayStatistics (FOR_ALL_CONNECTIONS) returns statistics for all connections in the pool

ReplayableConnection. clearReplayStatistics(StatisticsReportType) clears replay statistics – per connection or all connections

Runtime

TotalRequests = 1
TotalCompletedRequests = 1
TotalCalls = 19
TotalProtectedCalls = 19

Replay

TotalCallsAffectedByOutages = 3
TotalCallsTriggeringReplay = 3
TotalCallsAffectedByOutagesDuringReplay = 0
SuccessfulReplayCount = 1
FailedReplayCount = 0
ReplayDisablingCount = 0
TotalReplayAttempts = 3
Case Study – Instance Outage

Application Continuity replays – application sees no errors

1. Instance outage *
2. Replay driver receives error/FAN and connects to another RAC instance
3. Application Continuity replays
4. Application continues and returns to client

* Similar for session exit without FAN.
Case Study – Public Network Down

Application Continuity replays – application sees no errors

1. Public Network Down
2. Replay driver receives FAN from survivor and connects to another RAC instance
3. Application Continuity replays
4. Application continues and returns to client

Service members down  
Oracle JDBC Application  
Service members up  
RAC Instance

RAC Instances 1-n
Case Study – Site Down

Application Continuity replays – application sees no errors

1. Site or database down
2. FSFO observer waits FastStartFailoverThreshold
3. FSFO observer automated failover
4. Replay driver receives FAN from secondary site and connects to another RAC instance*
5. Application Continuity replays
6. Application continues and returns to client

Oracle JDBC Application

Database 1
- Primary
- DG Standby

Database 2
- DG Standby
- Primary

* Tuning tip: Set RETRY_COUNT and RETRY_DELAY to prevent errors for incoming connection requests
# Recommended Patches

<table>
<thead>
<tr>
<th>Component</th>
<th>Bug #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java Net</td>
<td>19154304</td>
<td>RETRY_COUNT did not include services</td>
</tr>
<tr>
<td>JavaNet</td>
<td>19000803</td>
<td>Provide TRANSPORT_CONNECT_TIMEOUT</td>
</tr>
<tr>
<td>RDBMS</td>
<td>19152020</td>
<td>PMON fast cleanup</td>
</tr>
<tr>
<td>RDBMS</td>
<td>19174056</td>
<td>Hang Manager extension</td>
</tr>
<tr>
<td>WebLogic</td>
<td>19587233</td>
<td>FAN and Application Continuity + JTS integration</td>
</tr>
<tr>
<td>WebLogic</td>
<td>20907322</td>
<td>FAN Autoons support</td>
</tr>
</tbody>
</table>
Lessons Learned

• Return connections to the connection pool between requests.

• Set http_request_timeout to allow the replay to occur

• Set REPLAY_INITIATION_TIMEOUT, RETRY_COUNT, and RETRY_DELAY

• Use mutable values. Think of mutables in terms of delayed execution.

• If the application sets values after creating a connection outside the application – repeat these settings in the callback.

• If the application is using XA datasource – check why. Most apps do not need XA.

• If testing and using V$instance etc, put in the callback to prevent mismatch as replay.
Transaction Guard

Unplanned outages should be hidden from applications
Transaction Guard
First RDBMS to preserve **COMMIT** Outcome

Reliable transaction outcome after outages

- Allows applications to deal with failures and timeouts correctly
- Without Transaction Guard, retrying can cause logical corruption
- Application Continuity uses Transaction Guard
- API available with JDBC-thin, OCI/OCCI, ODP.NET
How Transaction Guard Works

Oracle 12c Drivers

authenticate

........

........

COMMIT;

<get a new session>

Force commit outcome

COMMITTED?

COMPLETED?

Oracle 12c Database(s)

assign LTXID

start transaction

Session

New Session

Same DB Image

Preserve & Return

COMMIT OUTCOME

GET_LTXID_OUTCOME

Error or timeout

SQL, PL/SQL, RPC

COMMIT

LTXID

Oracle 12c Drivers

Oracle 12c Database(s)

Time
Transaction Coverage

**Inclusions**

Local
Commit on Success (auto-commit)
Distributed and Remote
DDL, DCL, parallel DDL
PL/SQL with embedded COMMIT
PL/SQL with COMMIT as last call
Read-only (allowed for)

**Exclusions**

XA in 12.1
Active Data Guard with database links used to commit at primary
Database Target - Coverage

**Inclusions 12.1**

- Single Instance Oracle RDBMS
- RAC One Node
- Real Application Clusters
- Data Guard
- Active Data Guard
- Multitenant including unplug/plug
  plus Transparent Application Failover (pre-integrated)

**Exclusions Database Failed Over To -**

- Logical Standby
- PDB Clones
- Golden Gate and third party replication
Forcing Commit Outcome

DBMS_APP_CONT.GET_COMMIT_OUTCOME forces the commit outcome, returning -

• COMMITTED
  – TRUE the user call executed at least one commit
  – FALSE the user call is uncommitted and stays that way

• USER_CALL_COMPLETED
  – TRUE the user call ran to completion.
  – FALSE the user call is not known to have finished
    e.g. use if app expects return data – e.g. commit on success, commit embedded in PL/SQL
Exceptions

• SERVER_AHEAD
  – the server is ahead of the client.
  – the transaction is an old transaction and must have already been committed

• CLIENT_AHEAD
  – the client is ahead of the server.
  – This can happen if the server has been flashed backed or using commit nowait

• ERROR
  – During processing an error happened.
Use Case - Unambiguous Outcome

Database session outage

FAN aborts dead session FAST
Application receives an error
Get last LTXID from dead session
Obtain a new database session

// Force commit outcome

execute DBMS_APP_CONT.GET_LTXID_OUTCOME using last LTXID

If committed then {
    process committed ; // let user or app know it committed
    if user_call_completed then application may continue
    else application may not be able to continue}

Else process uncommitted // let user know its safe to resubmit or resubmit automatically
What NOT to do – assume it did not commit

Connection jdbcConnection = getConnection();
boolean isJobDone = false;
while(!isJobDone) {
    try {
        // apply the raise (DML + commit):
        giveRaiseToAllEmployees(jdbcConnection, 5);
        // no exception, we consider the job as done:
        isJobDone = true;
    } catch (SQLRecoverableException recoverableException) {
        // On SQLRecoverableException, retry until isJobDone is true.
        try {
            jdbcConnection.close();
        } catch (Exception ex) {} // ignore any exception
        // Now reconnect so that we can retry:
        jdbcConnection = getConnection();
    }
}
What **NOT** to do – continued

```java
void giveRaiseToAllEmployees(Connection conn, int percentage) throws SQLException {
    Statement stmt = null;
    try {
        stmt = conn.createStatement();
        stmt.executeUpdate("UPDATE emp SET sal=sal+(sal*"+percentage+"/100)");
    } catch (SQLException sqle) {
        throw sqle;
    } finally {
        if(stmt != null)
            stmt.close();
    }
    // At the end of the request commit the changes:
    conn.commit();  // commit can succeed but return is lost
} .... (continued)
```
Solve with Transaction Guard - JDBC

Connection jdbcConnection = getConnection();
boolean isJobDone = false;
while(!isJobDone) {
    try {
        // apply the raise (DML + commit):
        giveRaiseToAllEmployees(jdbcConnection, 5);
        // no exception, the procedure completed:
        isJobDone = true;
    } catch (SQLRecoverableException recoverableException) {
        // Retry only if the error was recoverable.
        try {
            jdbcConnection.close(); // close old connection:
        } catch (Exception ex) {}
    } catch (Exception ex) {
    }
    Connection newJDBCConnection = getConnection(); // reconnect to allow retry
    // Use Transaction Guard to force last outcome: committed or uncommitted
    LogicalTransactionId ltxid = ((OracleConnection)jdbcConnection).getLogicalTransactionId();
    isJobDone = getTransactionOutcome(newJDBCConnection, ltxid);
    jdbcConnection = newJDBCConnection;
}
catch(Exception ex)
{
    OracleLogicalTransaction olt = con.OracleLogicalTransaction;
    olt.GetOutcome();  // obtains new connection

    if (!olt.Committed)  // guaranteed uncommitted
    {
        // safe for application or user to resubmit here
    }
    else
    {
        // transaction committed
        // test for completion – This part is not needed for top level commit, and when states are not needed
        if (olt.UserCallCompleted)
        {
            // return committed status
        }
        else
        {
            // return committed status - and warn that return states are unavailable
        }
    }
}
Required if using TAF Basic or TAF SELECT

• TAF handles Transaction Guard for OCI and ODP.NET apps
  – Set a boolean at connect and in TAF callback for TGenabled

  ```java
  catch (Exception ex)
  {
    //ONLY resubmit for the listed TAF errors and ONLY when TG is enabled
    if (TGenabled && (ex.Number == 25402 || ex.Number == 25408 || ex.Number == 25405 ))
    {
      // application may cleanup, then rollback and re-submit the current transaction
    }
    else
    {
      // handle the error as before; do not resubmit
    }
  }
  
  Refer MOS 2011697.1
## Use Case - Application Resubmits until Committed

<table>
<thead>
<tr>
<th>Force Commit Outcome</th>
<th>Application Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recoverable error occurs</td>
<td>Obtain <strong>LTXID-A-n</strong>, Get a new session, Execute <strong>GET_LTXID_OUTCOME</strong></td>
</tr>
<tr>
<td><strong>COMMITTED and COMPLETED</strong></td>
<td>Return committed and continue</td>
</tr>
<tr>
<td><strong>COMMITTED AND NOT COMPLETED</strong></td>
<td>Return committed, some apps cannot continue</td>
</tr>
<tr>
<td><strong>UNCOMMITTED</strong></td>
<td>Resubmit with a new session with <strong>LTXID-B-0</strong></td>
</tr>
<tr>
<td>Recoverable error</td>
<td>Obtain <strong>LTXID-B-n</strong>, Get a new session, Execute <strong>GET_LTXID_OUTCOME</strong></td>
</tr>
<tr>
<td><strong>UNCOMMITTED</strong></td>
<td>Resubmit with a new session with <strong>LTXID-C-0</strong></td>
</tr>
<tr>
<td><strong>COMMITTED and COMPLETED</strong></td>
<td>Return committed and continue</td>
</tr>
</tbody>
</table>
Server-side settings for Transaction Guard

• On Service
  – COMMIT_OUTCOME
    • Values – TRUE and FALSE
    • Default – FALSE
    • Applies to new sessions

• GRANT EXECUTE ON DBMS_APP_CONT TO <user>;}
Transaction Guard – Key Takeaway

First RDBMS to preserve commit outcome

- Users should not see misleading errors when a transaction really did commit.
- Driver receives an LTXID at authentication and on every commit.
- Once the commit outcome is returned, the result never changes.
- Safe for applications and mid-tiers to return success or resubmit themselves.
Success Stories
Out of the Box
## Unplanned Failover with Application Continuity

**WebLogic Active GridLink and Real Application Clusters**

<table>
<thead>
<tr>
<th>BEFORE</th>
<th>AFTER</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1.</strong></td>
<td>DB 11gR2+WLS Generic DS Error</td>
</tr>
<tr>
<td><strong>2.</strong></td>
<td>DB 11gR2+WLS Generic DS TIMEOUT 900s (TCP keep-alive)</td>
</tr>
<tr>
<td><strong>3.</strong></td>
<td>DB 11gR2+WLS Generic DS Error AP wait time:30s</td>
</tr>
<tr>
<td><strong>4.</strong></td>
<td>DB 11gR2+WLS Generic DS Hang AP wait time:minutes</td>
</tr>
</tbody>
</table>

1. **Instance down**
2. **Public network down**
3. **Interconnect down**
4. **Background process hang**

Diagram:
- **Oracle RAC & AppCont**
- **WebLogic**

---

Copyright © 2014 Oracle and/or its affiliates. All rights reserved. |
## Planned Failover with FAN

**WebLogic Server Active GridLink, RAC and Data Guard**

<table>
<thead>
<tr>
<th>DBA Operation</th>
<th>Maintenance</th>
<th>Result</th>
<th>Time to Drain all Sessions</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAC rolling</td>
<td>PSU apply using opatch</td>
<td>No errors to application</td>
<td>5s</td>
</tr>
<tr>
<td>RAC rolling</td>
<td>Instance parameter change</td>
<td>No errors to application</td>
<td>7s</td>
</tr>
<tr>
<td>Data Guard switchover</td>
<td>Site maintenance</td>
<td>No errors to application</td>
<td>29s</td>
</tr>
<tr>
<td>Data Guard switchover</td>
<td>Site maintenance fallback</td>
<td>No errors to application</td>
<td>25s</td>
</tr>
</tbody>
</table>
## Planned and Unplanned Failover

**RAC One Node, IBM WebSphere, Universal Connection Pool**

<table>
<thead>
<tr>
<th>Maintenance</th>
<th>Result</th>
<th>Time allowed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planned with FAN + Net</td>
<td>No errors to application</td>
<td>4 hours</td>
</tr>
<tr>
<td>Unplanned with Application Continuity + Net</td>
<td>No errors to application</td>
<td>10 minutes</td>
</tr>
</tbody>
</table>
## Runtime, Planned, & Unplanned

**ODP.NET Unmanaged Provider, RAC, and Data Guard**

<table>
<thead>
<tr>
<th>Database Method</th>
<th>Client Method</th>
<th>Example</th>
<th>Result</th>
<th>Time to Drain Sessions</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAC rolling upgrade/change</td>
<td>Drain with FAN + TNS</td>
<td>PSU / CPU</td>
<td>No errors to application</td>
<td>5s</td>
</tr>
<tr>
<td>Data Guard Switchover</td>
<td>Drain with FAN + TNS</td>
<td>Standby first PSU/CPU</td>
<td>No errors to application</td>
<td>25s</td>
</tr>
<tr>
<td>RAC Failover</td>
<td>Failover with FAN + TNS + TAF SELECT</td>
<td>Node outage</td>
<td>Errors for transactions</td>
<td>5s</td>
</tr>
<tr>
<td>Data Guard Failover</td>
<td>Failover with FAN + TNS + TAF SELECT</td>
<td>Site outage</td>
<td>Errors for transactions</td>
<td>-</td>
</tr>
</tbody>
</table>
For Developers: Application Continuity offloads the challenging work of transaction resubmission during failure events, allowing developers to focus on functionality.

Christo Kutrovsky – ATCG Principal Consultant, Oracle ACE

For Enterprise Architects: Application Continuity is a major step towards the holy grail of a continuously available, consistent, and highly performing database cluster

Marc Fielding – ATCG Principal Consultant, Oracle
The combinatorial solution with Application Continuity, Real Application Clusters, Data Guard, WebLogic Server Active GridLink and NEC hardware and middleware enables us to provide incredibly high available system for our Mission Critical customers. This solution will become our primary solution for cloud and big data areas.

Yuki Moriyama
Senior Manager, NEC Corporation
Safe Harbor Statement

The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle’s products remains at the sole discretion of Oracle.