Do a Data Guard switchover without your applications even knowing

Marc Fielding

September 2015

Succeeding at failure

Minimizing downtime is a constant challenge of Oracle practitioners. Planned downtime is scheduled in advance, and includes scheduled deployments, upgrades, and other maintenance tasks. This paper will focus on the other kind of downtime: the unplanned downtime that can have major impacts on systems and users.

Although we aspire to make systems as reliable as possible, it is impossible to eliminate failures entirely. So we must endeavor to make systems degrade gracefully when such failures happen. It requires a good understanding of what types of failure can happen, and what is the most reliable and elegant way to handle each.

Many applications deal with failure by printing out descriptive error messages. These can be invaluable to system developers to understand the cause of the failure and to address its root causes. To users, however, such error messages are of little use. Furthermore, error messages can leak confidential information about system architecture and internals, and create unnecessarily negative perceptions about the reliability of the system as a whole.

Rather, an elegant application should attempt to avoid sending error message to users where possible. It should be designed to tolerate failures with little or no user-visible impact.

A highly available database platform like Oracle Data Guard can be designed with excess capacity in case of failure or planned maintenance. Or in an online retail system, while the capability to order may be critical, history of former of orders or product recommendations are not. Determination of criticality of errors should be immediate, and not reliant on long timeouts from loosely coupled components. Applications must anticipate failure scenarios and be pre-programmed to make decisions based on the availability of external services.

Architecting for availability

When considering a highly-available design for an Oracle-based system, let us first consider implementing an availability framework entirely at the application level. In this example, we'll consider an application that renders product detail pages using data from an external recommendation service.

To avoid relying on long timeouts, such a service must be actively monitored, and state stored centrally. Application components can then query state from this central service before making a request to a known-failed component. Such a service must itself be highly available, low-latency, and accessible from all necessary application components. The active monitoring service must detect when a system has failed, or even just become to perform poorly, and flag availability status. Then callers of the recommendation service can make proactive decisions, such as temporarily skipping recommendation data while maintaining the functionality of the rest of the system.

High availability with Oracle Data Guard

Oracle Data Guard is designed to improve database availability by maintaining an up-to-date standby database, ready to take over the role of the primary database should an incident occur. Data guard features like standby redo logs and real-time apply help to reduce both downtime and data loss. Furthermore, to reduce the likelihood of a single incident affecting both primary and standby databases, the standby database can be located in a geographically-separated facility.

During a role change, however, an availability gap exists: how do we handle database sessions actively doing work? If a session has already modified data, and encounters a database failure, it must reconnect to the new primary database. And if such data changes are not yet committed, they are automatically rolled back. Applications therefore receive an error message from the database, and must manually reconstruct state. Almost all applications simply show an error message to the user, the exact goal we are hoping to avoid.

Prior to Oracle Database 12c, one workaround would be to add retry logic at the application. Such an application-level failure handling system would need to know the beginning and end of database transactions, and log pending changes independent of the database. Using process memory is an obvious approach, but what if such changes cannot fit in memory? The failure handling system must record the order of events, and respond to a database failure. If a failure occurred during a commit call, however, how can we know if the commit happened or not? An additional database table would be required to track a unique transaction ID, and to query after failure. Such a system effectively amounts to re-implementing Oracle's transaction layer!

Failure management in Oracle 12c

Oracle 12c has implemented features to address the transaction gap, with a goal of making database failure entirely transparent to applications. These features make failures appear to be abnormally slow requests, and avoiding error messages.

Transaction Guard is a key enabling technology. It provides a reliable method of determining of a previous commit happened or not. Such a mechanism is necessary to determine to avoid already-committed transactions being replayed again, and the associated potential for logical data corruption. The implementation is integrated with Oracle's core transaction-handling layer, and assigns a unique logical transaction ID (LTXID) to every new session; modifies this value at commit and returns changes to the driver as part of the ordinary acknowledgement message. A list of known LTXIDs is maintained over a 24-hour rolling window. After a failure, a call can be made to determine if a transaction was committed or rolled back. When Transaction Guard returns a commit outcome it stays that way: committed or uncommitted.

Application Continuity in action

An Application Continuity-aware configuration uses Transaction Guard to know if the last user submission committed or not. Application Continuity checks for edge cases where application replay is not possible (see further down for some examples). Full SQL statement details, including the statement, environment, bind variables, and other state are controlled by Application Continuity to ensure that if replay is required the correct state is restored, as if no outage had occurred. Replay is successful if the same user visible results can be restored.

When a failure occurs, the replay driver receives a failure notification. The replay driver reconnects to the new

primary database. Then Application Continuity performs sanity checks to verify that is the correct database and able to safely receive replay requests. Next, Transaction Guard is invoked to determine the state of in-flight transactions. If uncommitted changes exist, they are replayed under the direction of Application Continuity at the 12c database server. Each change is validated to ensure that user-visible results match. Following a successful replay, Application Continuity restores the session state, and control is passed back to the application.

System requirements

To run Application Continuity, both database server and driver must be running version 12c or higher. The database must be licensed for either Real Application Clusters or Active Data Guard, though these features do not need to be in use. As of Oracle 12.1.0.2, Application Continuity is used by the JDBC thin driver only, including WebLogic Server data sources.

To avoid situations when committed data changes are written to redo but lost due to communication failure with the standby, data guard should be configured in maximize availability or maximize protection modes, not maximize performance. The effect is to set the affirm parameter to the log destination, so that database sessions wait for the standby to receive redo before acknowledging commits. The exception is in maximize availability mode, where complete unavailability of the standby database does not prevent commits from happening, which would have the effect of exchanging one availability problem for another. When using maximize protection mode, a minimum of two standby databases should be configured in the addition to the primary, in order to mitigate these availability effects.

Oracle Data Guard should be configured with standby redo logs, and real-time apply can help speed up role transitions.

At the application level, there are a few features not supported by Application Continuity replay. One is JDBC concrete classes, as was implemented in Oracle's original 1995-era JDBC driver. Such calls are identified by oracle.sql rather than oracle.jdbc calls, and have continued to be supported until officially deprecated in Oracle 11.2.0.3. To support Application Continuity, such calls will need to be reimplemented using oracle.jdbc interfaces. Fortunately, these calls are rarely seen in current applications.

Client-side connection pools can complicate replay, as queries belong to entirely different requests can use the same database session. The database needs to know where requests begin and end. The request boundaries that Application Continuity uses are integrated into the Universal Connection Pool (UCP) and WebLogic Server Active GridLink connection pools, allowing it to understand where requests begin and end with these connection pools. If using application servers with other connection pools, such as WebSphere or plain Apache Tomcat, it is possible to add request boundaries manually through beginRequest and endRequest calls, though implementing UCP-based connection pooling is typically much easier.

Application Continuity is automatically replaying inflight transactions and as such automatically excludes situations that are unsafe for replay, such as:

- The instance default service – this service is for administrators' use, not for applications
- Global system changes that affect state system-side, like alter system or alter database. Session-level changes with global impact like dropping tablespaces are similarly excluded.
- Replay on any database that has lost data, including RESETLOGS operations, such as flashback

database, or incomplete media recovery

- With Active Data Guard, read/write database links back to the primary
- GoldenGate or logical standby target databases

The current version (12.1.0.2) of Application Continuity does not support X/Open (XA) transactions or third-party replication tools.

Mutable values

Mutable values are an interesting case when dealing with any type of replay, be it real application testing replay or application continuityApplication Continuity. They refer to operations returning different results on replay, even if called with the same arguments. The canonical examples are the sysdate and systimestamp functions, as well as sequence values. The replay driverApplication Continuity tracks the originally returned values during initial execution, and substitutes those values during replay. To allow such mutable values to be modified on replay, new privilege, "keep", must be granted. The grant can either be system-wide for sequences, date time, or sysguid, or applied to specific sequences.

Wrapping up

Application continuity Continuity fills in one of the largest remaining availability gaps in the Oracle database, allowing applications to handle loss of entire database servers without complex and unreliable application-side transaction handling. When combined with the existing high-availability capabilities of Oracle Data Guard, we get a reliable way to handle disasters without application-visible downtime.

Acknowledgements

Many thanks to Jeremiah Wilton, who prepared much of the failure-handling part of this paper. Also a special thanks to Carol Colrain at Oracle for her knowledge and passion about the product, and her help in correcting my misunderstandings.

Additional resources

12c Application Continuity with Data Guard in action, Bertrand Drouvot, May 2015

Application Continuity with SwingBench, Dominic Giles, November 2014

Application Continuity with Oracle 12c, Carol Colrain, July 2014

Application Failover with Oracle Database 11g, Kuassi Mensah and Norman Woo, September 2010

Oracle Database 12c Application Continuity for Java, Kuassi Mensah, June 2013

Playing with Application Continuity in Oracle 12c, Martin Bach, December 2013

Transaction Guard with Oracle Database 12c, Carol Colrain, February 2015

Zero downtime: Hiding Planned Maintenance and Unplanned Outages from Applications, Carol Colrain, 2014

Contact information

Marc Fielding

twitter.com/mfild

linkedin.com/in/mfielding

fielding@gmail.com