Hochverfügbarkeit für den Anwendungsentwickler mit Oracle Datenbank 12^{c}

Autor: Sebastian Solbach, Oracle

Oracle 11^{g}R2 bietet viele Funktionalitäten, die Datenbank hochverfügbar und ausfallsicher zu gestalten. Oracle RAC und Data Guard sind dabei wichtige Technologien dies zu erreichen, beschränken sich in Ihrer Funktionalität allerdings auf die Datenbank. Aber was bringt einem Hochverfügbarkeit, wenn die Applikation damit nicht umgehen kann? Mit Oracle 12^{c} führt Oracle nun Transaction Guard und Application Continuity ein, um genau diese Lücke zu schließen.

In der Applikationsentwicklung gibt es ein grundlegendes Problem, dass sehr schwer zu lösen ist: Den Status einer sogenannten „In-flight“ Transaktionen nach einem Ausfall zu bestimmen, egal ob dieser Ausfall auf Basis eines Problems im Netzwerk oder in der Datenbankschicht verursacht worden ist. Neben den offensichtlichen Auswirkungen, wie unter Umständen den Neustart der Mid-Tier und der Frustration der Benutzer, weil die Applikation abgestürzt ist, bietet sich für den Entwickler die entscheidende Frage, wie er mit diesem Ausfall und der „in-flight“ Transaktion umgehen soll. Denn der Entwickler hat zumindest bis 12^{c} keine verlässliche Möglichkeit herauszufinden, ob seine Transaktion nun erfolgreich war, zurück gerollt worden ist oder noch schlimmer immer noch in „Bearbeitung“ in der Datenbank ist. Als Ergebnis dieses Dilemmas kann es im schlimmsten Fall zu doppelten Buchungen und einer logische Korruption kommen, weil Transaktionen nicht oder doppelt durchgeführt werden.

Genau genommen ist dies aber nicht ein Problem, sondern besteht aus mehreren grundlegenden Detailproblemen, die alle berücksichtigt werden wollen:

- Ausfall feststellen
- Wiederaufnahme im vorgegebenen Zeitfenster
- Fehlerkorrektur
- Transaktionsstatus herausfinden
- Weiterführung aktueller Transaktionen

Status vor 12^{c}

Mit 11^{g}R2 sind die ersten 3 Punkte zumindest teilweise adressiert. So ist es mit Hilfe von Fast Applikation Notification in Sekundenbruchteilen möglich, von einem Vorkommnis auf der Datenbankebene informiert zu werden, egal ob eine geplante Auszeit, das Auftreten eines Problems oder die Verfügbarkeit zusätzlicher Rechenkapazität die Ursache ist. Über FAN können die unterschiedlichen Treiber (JDBC/OCI/.NET) und deren Connection Pools informiert werden und die entsprechende Aktion, wie z.b. das „Draining“ des Connection Pools automatisch initiiert werden. Hierbei stellt bei 12^{c} der Oracle Notification Server für alle Funktionen den Nachrichtenservice bereit, diese Informationen zu verbreiten.
Real Application Clusters und Data Guard sind die Technologien den Datenbankservice für die Applikation so schnell wie möglichst wieder zur Verfügung zu stellen. Je nach gewählter Technologie unterscheiden sich aber die Zeit zur Wiederaufnahme einer Aktion.

Für den Applikationsentwickler wurde auch schon in 11^{g} das Error Handling erheblich einfacher, da die Applikation nun nicht mehr eine eigene Liste mit Fehlercodes mit-pflegen musste. Anstelle der unterschiedlichen Fehlercodes rückt nun nur noch die Information, ob der entstandene Fehler „Recoverable“ ist oder ob ein schwerwiegenderes Problem vorliegt. Auskunft darüber gibt die JDBC SQLRecoverablException bzw. das OracleException.IsRecoverable Property in OCI.

Die übrigen 2 Probleme löst nun 12^{c} mit Transaction Guard und Application Continuity.

Basis Setup der Datenbank für Transaction Guard und Application Continuity

Einige Grundlegende Vorbereitungen müssen auf der Datenbank getroffen werden, damit der Entwickler von den neuen Funktionalitäten profitieren kann. Die wichtigste grundlegende Änderung ist, dass man zur Verbindung mit der Datenbank einen eigenen Service verwenden sollte und auf keinen Fall die SID oder den Default Datenbank Service. Der DBA gibt diesem Service einige Parameter mit, die für die Aktivierung von Transaction Guard und Application Continuity zuständig ist, wie hier am Beispiel für einen RAC Cluster mit dem Befehl srvctl:

```
srvctl modify service -db ORCL -service APPCON
-failovertype TRANSACTION -replay_init_time 300
-failoverretry 30 -failoverdelay 3
-notification TRUE -commit_outcome TRUE
```

- COMMIT_OUTCOME bestimmt dabei ob die Datenbank für die einzelnen Transaktionen eine eindeute logische TransaktionsID speichert und ist die grundlegende Einstellung für Transaction Guard.
- RETENTION_TIMEOUT bestimmt dabei die Zeit in Sekunden, wie lange diese Informationen in der Datenbank vorgehalten werden - im Default sind das 24 Stunden, was für die meisten Applikationen absolut ausreichend sein sollten.
- AQ_HA_NOTIFICATIONS steht für die Übermittlung der FAN Events auch an OCI wie .NET Clients, da die Implementation dieser Funktionalität in 12^c keinen Overhead mehr erzeugt, sollte der Parameter eigentlich immer angeschaltet sein.
- FAILOVER_TYPE aktiviert mit dem Wert TRANSACTION dass Application Continuity möglich ist.
- REPLAY_INITIATION_TIMOUT gibt die Zeit in Sekunden für Application Continuity vor, wie lange ein Replay dauern darf.
- FAILOVER_RETRIES und FAILOVER_DELAY sind Angaben, wie oft und in welchen Zeitabständen der Client versuchen sollte, sich neu zu verbinden.

Des weiteren muss der DBA dafür sorgen, dass der Applikationsbenutzer die Informationen von der Datenbank über das Ergebnis eines Commits erfahren darf:

```
GRANT EXECUTE ON DBMS_APP_CONT TO <app-name>;
```

Transaction Guard

Transaktion Guard bietet zum ersten mal in der Geschichte der Datenbanken die Möglichkeit, dass Applikationen richtig auf Fehler reagieren können und deckt damit ein Fehlerszenario ab, welches bisher mit keiner Technologie wirklich gelöst werden kann: Angenommen eine Applikation committed zum Zeitpunkt t0 eine Transaktion und erhält eine Fehlermeldung. Auf Basis dieser Fehlermeldung (Not committed) entscheidet die Applikation das weitere Vorgehen und führt die Aktion nochmals aus.

Zum Zeitpunkt t1 kann die Datenbank die „hängende“Transaktion aus t0 doch noch abschließen. Damit taucht dieselbe Transaktion 2x in der Datenbank auf.

Dieses und andere Szenarien deckt nun Transaktion Guard ab. Dazu assoziiert das RDBMS zu jeder Transaktion eine eigene logische TransaktionsID (LTXID). Die LTXID ändert sich nur bei einem erfolgreichen COMMIT oder einem erfolgreichen ROLLBACK. Die LTXID wird an den unterstützen Treiber kommuniziert. Falls die Verbindung abbricht, wird der Status der Transaktion mit Hilfe der LTXID erneut abgefragt:

```
LogicalTransactionId ltxid = oldCon.getLogicalTransactionId();
OracleConnection newCon = getConnection();
CallableStatement cstmt =
newCon.prepareCall(GET_LTXID_OUTCOME);
```

Die Applikation erfährt nun entweder das Ergebnis der Transaktion (Committed oder Rollback), gleichzeitig verhindert das RDBMS dass die Transaktion, sollte diese noch nicht im RDBMS angekommen sein, committed werden kann.

Damit können alle Arten von Commit Modellen, wie Autocommit, Embedded PL/SQL, DDL und Paralleles DDL abgesichert werden, egal ob diese OCI, .NET oder JDBC verwenden. In 12.1 sind lediglich XA Transaktionen und R/W Links bei Active Data Guard ausgeschlossen.

Application Continuity

Transaction Guard erfordert immer noch, dass der Entwickler die Funktionalität in die Applikation einbaut. Unter JDBC Thin geht die Integration von Transaction Guard noch einen Schritt weiter: Mit Hilfe von Application Continuity werden Applikationen befähigt, ohne Applikationsänderung einfach einen Ausfall zu tolerieren.
Allerdings basiert das Ganze auf einer Art Wiederholungsstrategie, d.h. der JDBC Thin Treiber merkt sich die einzelnen Statements bis zum einem Commit und wird diese im Fehlerfall wiederholen. Transaction Guard hilft dabei zu definieren, ob überhaupt wiederholt werden muss.

Entscheidend ist dabei, was der Treiber hier als einen Datenbank Request betrachtet. Im Normalfall ist das alles zwischen einem getConnection und einem Commit, gefolgt von einem conn.close().

Applikation Continuity besteht somit aus 3 Phasen:

1.) Zur Laufzeit werden die einzelnen Datenbankrequest aufgezeichnet, ermittelt was bzw. was nicht wiederholt werden kann und die einzelnen Ergebnisse werden inkl. Bind Variablen zur späteren Validierung im Treiber zwischengespeichert.
2.) Einer Reconnect Phase im Fehlerfall, in der mit Hilfe von Transaction Guard geprüft wird, ob ein Wiederholung überhaupt möglich/notwendig ist. Das passiert unter Berücksichtigung von Timeouts und ob die Zieldatenbank/Instanz für eine Wiederholung überhaupt geeignet ist.
3.) Der Replay Phase, in der gespeicherte Aufrufe neu ausgeführt werden und die entsprechenden Ergebnisse der einzelnen Aufrufe validiert werden.

Zur Konfiguration von Application Continuity ist lediglich die Verwendung der Replay Enabled Data Source notwendig (neben den Verwendungen eines Services und den Vorbereitungen des DBAs!).

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

Leider lassen sich nicht alle Aufrufe so ohne weiteres neu Abspielen. Dabei gibt es einige Globale Restriktionen, Einschränkungen bei einzelne Requests oder ganz generelle Einschränkungen bezüglich des Ziels. Zu den globalen Restriktionen zählt neben der Verwendung des falschen, bzw. Default Services der Datenbank bzw. der PDB, XA Transaktionen und die Verwendung einiger JDBC Klassen aus dem oracle.sql Package. Verständlich hingegen ist, dass sogenannte „restricted calls“, wie ein „alter system“ und „alter database“ nicht wiederholt werden sollten. Auch funktioniert Application Continuity nur in derselben Datenbank, d.h. RAC und Active Data Guard, nicht jedoch bei Golden Gate und Logical Standby, da diese als „andere“ Datenbank andere LTXIDs erzeugen würden.

Neben diesen Einschränkungen, die das Replay generell verhindern, gibt es auch einzelne Aufrufe, die „nur“ nicht wiederholt werden. Dazu gehören währen der Laufzeit Transactionen nach einem Commit aber innerhalb eines Requests. Als Workaround müssen hier die Grenzen manuell vom Entwickler festgelegt werden. Außerdem hat der Entwickler selbstverständlich auch die Möglichkeit für einzelne Aufrufe mit Hilfe der disableReplay API eine erneute Ausführung einzelner Funktionen zu verhindern. Während der Reconnect Phase kann ein Replay verhindert werden, weil der aufgetretene Fehler nicht „recoverable“ ist oder weil sich einfach innerhalb des entsprechenden Zeitfensters nicht neu verbunden werden kann. Als letztes könnte die Wiederholung noch daran scheitern, dass sich Ergebnisse z.B. eines Selects geändert haben.

Ist meine Applikation für AC geeignet oder sind manuelle Anpassungen erforderlich?

Jede JDBC Thin Applikation kann von Application Continuity profitieren, allerdings bedeutet das im schlimmsten Fall, dass die Grenzen für den Replay jeweils manuell festgelegt werden müssen. Es gibt aber auch viele Fälle, in denen absolut keine manuelle Änderung notwendig ist, wie Abbildung 1 zeigt:

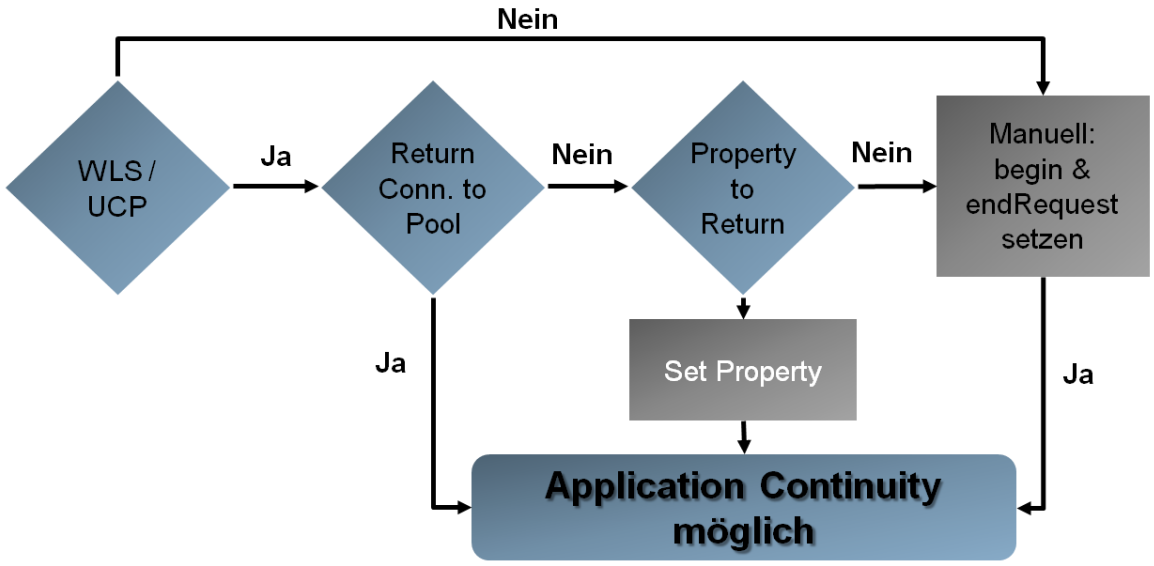

Abbildung1: Geeignet für Application Continuity

Applikationen, die auf WebLogic Server laufen oder den Oracle Universal Connection Pool verwenden, können direkt von AC profitieren, wenn Sie Verbindungen nach einem Request wieder an den Connection Pool zurückgeben. Selbst wenn die

Applikation aufgrund von Performanceoptimierungen den Session State extern setzt und somit nicht unbedingt die Connection so einfach zurückgeben kann, sind hier einfache Workarounds möglich. Bei WebLogic würde man das sogenannte „Connection Labelling“ hierzu verwenden, welches implizit von AC unterstützt ist oder für das Setzen des Properties auf einen Callback von AC ausweichen.

Eine letzte Hürde gibt es noch zu nehmen, wenn Funktionen wie SYSDATE(), SYSGUID() oder Sequences verwendet werden. Im Normalfall würde bei einem Replay die Sequence erhöht oder das aktuelle Datum zurückgegeben, nicht das Datum der letzten Transaktion. Somit würde ein Replay nicht funktionieren, da bei der Validierung des Ergebnisses ein Unterschied auftritt und das Replay abbrechen würde. Hierfür muss der Applikationsuser ein „GRANT KEEP TIME“ oder „GRANT KEEP SEQUENCE“ besitzen, damit bei einem Replay die alten Informationen verwenden werden dürfen.

Fazit

Oracle 12^c kommt MAA (Maximum Application Availability) erheblich nahe, auch wenn noch nicht zu 100%. Allerdings bietet 12^c zum ersten mal, seit dem es Datenbanktechnologien gibt, eine verlässliche Möglichkeit Transaktionen wieder aufzunehmen und weiterzuführen.

Links

- Deutsche DBA Community: https://blogs.oracle.com/dbacommunity_deutsch/
- Application Continuity on OTN http://www.oracle.com/technetwork/products/clustering/ac-overview-1967264.html
- Whitepapers:
    - Transaction Guard: http://www.oracle.com/technetwork/database/database-cloud/private/transaction-guard-wp-12c-1966209.pdf
    - Application Continuity: http://www.oracle.com/technetwork/database/database-cloud/private/application-continuity-wp-12c-1966213.pdf

- Oracle 12c Dokumentation (hier Universal Connection Pool): http://docs.oracle.com/cd/E16655_01/java.121/e17659/app_cont.htm#JJUCP8254

Kontakt:

Sebastian Solbach

sebastian.solbach@oracle.com