Hochverfügbarkeit mit Replikation – Logisch?

Sebastian Winkler
CarajanDB GmbH
Erftstadt

Schlüsselworte

Hochverfügbarkeit, high availability, HA, Replikation, Physical Standby, Dbvisit

Hochverfügbarkeit

Die gängigsten Lösungsansätze beim Thema Hochverfügbarkeit im Oracle Umfeld lauten, RAC, Data Guard und GoldenGate. Mit Hochverfügbarkeit (englisch high availability, HA) bezeichnet man die Fähigkeit eines Systems, einen unterbrechungsfreien Betrieb einer Anwendung, auch im Fehlerfall, mit einer möglichst hohen Wahrscheinlichkeit zu gewährleisten. Welche Lösung überhaupt zum Einsatz kommen sollte und den Anforderungen entspricht gilt es als erstes zu klären. Denn soll ein Ausfall der Anwendung sprich der Datenbank oder soll ein möglicher Datenverlust abgesichert werden? Beide Anforderungen führen zu den unterschieden zwischen den zuvor genannten Lösungen. Doch was wenn man nach einer Alternative ohne den Einsatz von Enterprise Edition sucht? Wenn die Features der Standard Edition für die primären Anforderungen vollkommen ausreichend sind? Zunächst erscheint eine Lösung abseits der umfangreichen Werkzeuge die Oracle zur Verfügung stellt schwierig. Aber wenn man streng kalkuliert und die Mittel des Unternehmens möglichst effektiv eingesetzt werden sollen, kann man auch schon mal über den Tellerrand hinweg schauen und findet Lösungen von verschiedenen Drittanbietern. Neben der bekannten Lösung von Quest (Dell Software) mit SharePlex hat vor allem Dbvisit mit seiner Standby Lösung von sich reden gemacht. Seit längerem hat Dbvisit mit Replicate auch eine eigene Replikationslösung als Alternative zu Oracle Streams im Portfolio. Eine Alternative die man sich vor allem ansehen sollte wenn man das Feature bereits im Einsatz hat. Gerade vor dem Hintergrund, dass Oracle Streams den Status „deprecated“ zugewiesen hat. Was bedeutet, dass das Feature noch verfügbar ist, aber in Zukunft nicht mehr weiterentwickelt und unterstützt wird. Schade. Der Verweis auf GoldenGate führt hier wieder zu hohen Lizenzkosten und Features die vielleicht gar nicht gebraucht werden.

Warum eine Replikation?

Eine redundante Datenhaltung, abseits von gespiegelten Platten kann über eine Replikation, entweder logisch oder als „physikalische“ Standby erfolgen. Die Unterschiede liegen in der Nutzbarkeit des Replikats. Während eine Physical-Standby ausschließlich für den Fehlerfall aufgebaut wird, kann bei einer logischen Replikation die Zieldatenbank voll genutzt werden, zum Beispiel als Reporting Datenbank. Beim Einsatz der Oracle Enterprise Edition mit Active Data Guard und Logical Standby stehen gleich mehrere Lösungen für eine verlustfreie Replikation bei gleichzeitiger Nutzung des Replikates zur Verfügung. Die Herausforderung bei der Standard Edition ist jedoch ungleich höher. Lösungen, die auf ein Redolog-Shipping basieren, wie z.B. Dbvisit Standby erlauben zwar eine elegante und schnell zu realisierende Hochverfügbarkeitslösung, allerdings mit dem Nachteil, dass bei einem Fehler ein Datenverlust von mehreren Minuten (praktisch ca. 15 Minuten) in Kauf genommen werden muss und das Replikat während des „normalen“ Betriebs nicht genutzt werden kann. Daher lautet die wohl wichtigste Frage bei der Analyse einer Hochverfügbarkeitsumgebung mit der Standard Edition: Wie viel Datenverlust kann toleriert werden? Wenn die Anzahl der Transaktionen eher gering ist, kann eventuell eine Standby-Lösung in Frage kommen, bei der die Redolog-Dateien alle 5 Minuten übertragen werden – aber Vorsicht! Bei Batchläufen oder anderen größeren Aktionen kann

dies schnell zu einem Engpass werden. Das bedeutet, wenn der maximale Datenverlust unterhalb von 15 Minuten liegen muss, sollte von einer physikalischen Replikation abgesehen werden. Hier kann die logische Replikation Ihre Stärken ausspielen und den Datenverlust bei einem Failover auf ein Minimum reduziert werden.

Wie funktioniert die Replikation?

Bei der logischen Replikation erfolgt im Gegensatz zur Standby-Lösung kein Recovery der archivierten Redolog-Dateien, sondern die DML- und DDL-Befehle werden direkt aus den Redolog-Dateien wiederhergestellt und auf der Standby ausgeführt. Wie sieht der minimale Datenverlust nun aus? Dieser beschränkt sich also lediglich auf Transaktionsdaten, die bis zum Failover noch nicht von der Primärseite zur Replikationsseite verschickt wurden. Soll heißen alles was in den Redolog-Files gelandet ist und appliziert wurde ist auch auf der replizierten Datenbank. Bei der Übertragung der Daten gibt es zwei konzeptionelle Unterschiede: DELL SharePlex und Dbvisit Replicate übertragen die Daten, bevor das Commit erfolgt, während GoldenGate die Daten gemeinsam mit dem Commit überträgt. Im Fehlerfall könnte also ein einzelnes „Commit“ noch durchgeführt worden sein, während größere Transaktionen, die als Gesamtheit übertragen wurden, wahrscheinlich nicht erfolgreich sind.

Wo liegen die Herausforderungen?

Während eine Physical-Standby schnell aufgebaut ist und generell, neben einem Monitoring kein weiterer Aufwand erfolgt, handelt es sich bei einer logischen Replikation um eine eigenständige Datenbank, die separat verwaltet und bei produktiver Nutzung auch gesichert werden muss. Letzteres sollte kein unlösbares Problem sein. Viel schwieriger werden die Fragen wenn man weiß, dass Änderungen in den SYS und SYSTEM Schemas bei der logischen Replikation nicht erfasst werden. Gleiches gilt aus diesem Umstand heraus für ROWIDs. Weitere Problemstellungen können sich aus speziellen Datentypen die von der jeweiligen Replikationslösung nicht unterstützt werden bspw. XMLTYPE ergeben. Noch eine Aufgabe stellen Trigger, die jetzt zweimal feuern und Sequences die gerne als eindeutige IDs verwandt werden. Das Problem lautet, mit und ohne Replikation, dass eine ununterbrochene Zahlenfolge nicht garantiert werden kann, z.B. bei einem Rollback einer Transaktion. Trotz dieses Umstands verlassen sich viele Anwendungen für die Nummerierung von Rechnungen auf Sequences bzw. den Primary-Key der zwar garantiert eindeutig ist, aber nicht garantiert ununterbrochen fortlaufend. Bei einem Failover kann es hier zu Diskrepanzen kommen. Ein letztes Problem betrifft Konflikte, die entstehen können wenn Daten auf der Replikationsseite durch Constraints verhindern, dass Daten von der Primärseite eingefügt werden können. Dieses Problem kann aber nur auftreten, wenn Änderungen von Dritter Seite in der Replikationsdatenbank vorgenommen wurden. Hierfür muss ein entsprechender Zugriffsschutz gewährleistet werden. Die letzte große Herausforderung liegt darin, die Komplexität so niedrig wie möglich zu halten und ein praktikables System zu entwickeln, dass nicht nur die Anforderungen erfüllt sondern auch beherrschbar bleibt.

Wie sehen die Lösungen aus?

Bei allen Überlegungen rund um eine Hochverfügbarkeit bei Datenverlust darf man natürlich menschliches Versagen nicht außer Acht lassen und muss für diesen Fall auch hier eine belastbare Backup-Strategie vorhalten. Lösungsansätze und wie eine solche Replikationsumgebung mithilfe von Dbvisit Replicate aufgebaut werden kann, welche Rolle der RMAN noch spielen wird und wie groß der Unterschied zwischen einem Graceful Switchover und einem Failover sein kann, werden im Vortrag anhand eines realen Projektes präsentiert.

Projektanforderungen

Im vorgestellten Projekt wird zusammen mit einer Hardwaremigration von einer ehemals Oracle Enterprise Edition 11g mit Data Guard Umgebung auf eine Oracle Standard Edition 11g mit Dbvisit Replicate migriert. Somit waren für die Verantwortlichen Datenbankadministratoren und Entwickler gleich mehrere Baustellen zu bewältigen. Dieser Umbau folgte dem Leitgedanken, die Einsparungen zum einen in leistungsfähigere Hardware und zum anderen in ein verbessertes Backup-Konzept fließen zu lassen. Der folgende Teil beschränkt sich dabei lediglich auf Einrichtung, Probleme und Lösungen im Zusammenhang mit der Hochverfügbarkeitsumgebung. Auf Seiten der Datenbank haben wir ein Volumen von mehreren 100 GB Daten mit Zugriffen von internen und externen Mitarbeitern aus verteilten Standorten neben Deutschland aus Süd-, West- und Osteuropa. Dieser dann ein Windows-Server 2008 R2 der über 64 GB RAM mit 2 Octa-Core Prozessoren und SAS Festplatten mit 6 Gb/s Übertragungsrate zur Verfügung steht. Für die Hochverfügbarkeitsumgebung wurde der Server entsprechend ein zweites Mal in einem weiteren Rechenzentrum aufgebaut. Die Projektleitung wollte mit diesen Voraussetzungen eine HA-Umgebung schaffen, die den geringsten Datenverlust gewährleistet. Wie bereits beschrieben sind die Möglichkeiten mit einer Physical Standby Datenbank dabei auf einen gewissen Zeitfaktor beschränkt. Daher entschied man sich den Weg einer logischen Replikation zu gehen und wählte dafür Dbvisit Replicate als Drittanbieter-Produkt aus.

Einrichtung

Die ersten Tests fanden auf den zukünftigen Servern mit einer Kopie der Produktionsdatenbank statt. Die DBAs haben somit die Möglichkeit, mit der zukünftigen Produktionsumgebung alle möglichen Szenarien durchzuspielen und sich mit den Abläufen vertraut zu machen. Es empfahl sich zudem für die Zukunft eine Test-Umgebung bereitzustellen um weitergehende Änderungen an der Datenbankstruktur und eventuellen Auswirkungen auf die Replikation vorher durchspielen zu können.

Bevor die Replikation beginnen kann muss sichergestellt werden, dass auf beiden Seiten die gleiche Datenbank-Struktur und Inhalte bereitstehen. Da wir mit einer bereits gewachsenen Datenbank arbeiten, können wir dies am besten mit einem RMAN Duplicate umsetzen. Nach der Software-Installation auf der Standby-Seite klonen wir die Datenbank einfach 1:1 mit Hilfe von Oracle eigenen Mitteln. Damit haben wir Konflikte die beispielsweise durch ein manuelles aufsetzen der Datenbank und deren Strukturen sowie einem manuellen hineinladen der Daten entstehen können vermieden. RMAN macht es uns an dieser Stelle recht einfach und nach dem Erstellen und Mounten einer Instanz auf der Standby-Seite brauchen wir lediglich folgenden Befehl absetzen:

```
RMAN> DUPLICATE TARGET DATABASE TO 'prod' FROM ACTIVE DATABASE
NOFILENAMECHECK;
```

Nachdem die Datenbank vollständig geklont wurde, können wir mit der Installation und Einrichtung von Dbvisit Replicate beginnen. Die Installation wird auf beiden Seiten vorgenommen. Nach der Softwareinstallation verläuft die Konfiguration der Replikation in vier Schritten. Dabei erfolgt die Verwaltung und das Logging der Replikation über ein Schema, welches jeweils auf beiden Seiten für Dbvisit angelegt wird. Die Replikation selbst kann sowohl in eine Richtung als auch in beide Richtungen (2-way replication) eingerichtet werden. Wobei für uns, zur absoluten Konfliktvermeidung, nur die Replikation von der Primary in Richtung Standby in Frage kommt. Genau festgelegen können wir auch welche Tabellen und/oder Schemas wir replizieren wollen. Schließlich wird ein sogenannter MINE Process auf der Primary- und ein APPLY Process auf der Standby-Seite konfiguriert. Nach dem Starten der vorkonfigurierten Services läuft die Replikation bereits. Dazu generiert Dbvisit aus den Redo-Logs der Datenbank sogenannte PLOGs auf der Primary-Seite, schiebt diese über das Netzwerk auf die Standby-Seite und appliziert diese auf die Standby-Datenbank.

Kernprobleme und Lösungen

Wie bereits in den Herausforderungen beschrieben, müssen zu den typischen Problemen einer Replikationslösung auch für diese Umgebung individuelle Lösungen gefunden werden. Alle nachfolgenden Aussagen beziehen sich ausschließlich auf eine Replikation mit Dbvisit Replicate und können sich von anderen Lösung unterscheiden. Der erste Umstand dem wir uns gewahr sein müssen ist, dass es sich bei der replizierten Datenbank um eine eigenständige Datenbank handelt. Eigenständig heißt zunächst einmal, dass SYS und SYSTEM Objekte nicht repliziert werden. Dies bedeutet weiter, dass hier ein entsprechendes Monitoring stattfinden muss. Noch entscheidender ist, dass dadurch nicht jedes DDL unterstützt wird. Das Anlegen eines neuen Tablespaces beispielsweise nicht auf beiden Seiten automatisch erfolgt. Inzwischen repliziert Dbvisit grundsätzlich keine Trigger, Constraints, Sequences, ALTER DATABASE und ALTER SYSTEM Kommandos oder Datenbankstrukturen. Hierfür muss ein entsprechender Ablauf im Betrieb geschaffen werden, der dafür sorgt, dass derartige Änderungen auf beiden Seiten angewendet und auf Konformität hin überprüft werden. Ein weiteres Problem stellen spezielle Datentypen dar, die von Dbvisit Replicate nicht unterstützt werden und daher, sofern genutzt, anderweitig auf die replizierte Datenbank gebracht werden müssen. In diesem Projekt betraf das den XMLTYPE. Mit Hilfe von TOAD und einem DB Compare lassen sich Unterschiede an dieser Stelle am schnellsten aufdecken und je nach Bedarf auf die andere Seite anwenden.

Da die Entwickler der Datenbank eine ausgiebige Nutzung von Triggern pflegen, ist bereits bei den ersten Tests das Problem aufgetreten, dass diese nun auf beiden Seiten feuerten. Zu dieser Zeit wurden Trigger-Kommandos noch repliziert. Das bedeutet, dass ein durch einen Trigger ausgelöstes DML-Kommando auf der Primärseite, einmal durch die Replikation eingetragen wird und der Trigger selbst durch die Replikation ein zweites Mal auf der Standby-Seite ausgeführt wird. Durch den doppelten Eintrag bzw. den Versuch diesen durchzuführen, entsteht an dieser Stelle sofort ein Konflikt und die Replikation steht. Sie steht, weil die Replikation solange einen „retry“ durchführt, bis der Konflikt aufgelöst wurde. Dbvisit bietet dazu einen Conflict Handler der uns generell zwei Möglichkeiten gibt, einmal den Konflikt zu ignorieren und das Statement wird zurückgerollt oder ein overwrite und die Ziel-Daten werden unter Missachtung der WHERE-Klausel einfach überschrieben. Keine der Methoden gibt uns die Gewissheit, dass wir am Ende eine Datenkonformität haben und wir sicher sein können, dass auf beiden Seiten der gleiche Inhalt geschrieben wurde. Was für eine Test-Umgebung vielleicht tolerabel ist, muss für die Sicherung einer Produktionsdatenbank ausgeschlossen werden. An dieser Stelle wird also nochmal deutlich warum keinerlei Konflikte toleriert werden können. Die Lösung für das Trigger-Problem ist dann auch denkbar einfach. Da wir vollständigen Zugriff auf das Replikat haben, müssen wir vor Beginn der Replikation alle Trigger ausschalten.

```
SQL> ALTER TABLE <table_name> DISABLE ALL TRIGGERS;
```

Die letzte Herausforderung der wir uns stellen müssen ist das Umschalten im Falle eines Ausfalls. Der Hauptzweck unserer Hochverfügbarkeitsumgebung, der geringstmögliche Datenverlust wäre durch die Replikation realisiert. Das genügt aber nicht, denn das Replikat sollte im Zweifel als Produktiv-Umgebung nutzbar gemacht werden. Die Hürde die genommen werden muss heißt also Switchover und Failover. Als erste Maßnahme, um die Replikation vor ungewollten Zugriffen zu schützen, legen wir auf beiden Seiten einen gleichnamigen dynamischen Datenbank Service an, den wir über das DBMS_SERVICES Package steuern können. Damit wird der Zugriff der End-User Clients ausschließlich über diesen Service realisiert, den wir jederzeit starten und stoppen können. Auf Seiten der Clients sind beide Server eingetragen und eine Anmeldung kann nur auf der Produktionsseite erfolgen, auf der der Service gestartet wird. Im Falle eines Ausfalls der Produktion könnten wir also dann den Service auf der Replikationsseite freigeben, wenn wir sichergestellt haben, dass hier alle produktionsrelevanten Daten bereit stehen.

```
SQL> EXEC DBMS_SERVICE.START_SERVICE('PRODUSER');
```

Es gibt noch ein weiteres Problem, dass durch das Wesen unserer Replikation verursacht wird. Da Sequences nicht mit Repliziert werden, müssen diese vor dem freischalten für die Produktion an den aktuell höchsten Wert angepasst werden. Zur Sicherheit empfiehlt es sich den höchsten Wert zu ermitteln und noch bspw. 100 zu addieren um ganz sicher zu gehen, dass keine Konflikte entstehen. Dieser Umstand lässt dann sofort Diskussionen aufkommen, die in Richtung Buchhaltung und Finanzamt führen, die bspw. für Rechnungen eine unbedingt fortlaufende Nummer einfordern, andernfalls steht man mit einem Bein im Gefängnis. Das mag sein. Aber nicht die fortlaufende Nummer ist das Problem sondern Sequences die unter keinen Umständen eine fortlaufende Nummerierung garantieren. Ein einfacher Test mit einem Rollback zeigt, dass eine einmal verwendete Sequence weg ist und sofort eine „Lücke“ verursacht. Die Aufgabe an die Entwickler heißt an dieser Stelle also einen anderen Weg finden, denn eine Sequence ist nicht das geeignete Mittel eine fortlaufende Nummerierung zu garantieren.

Produktion

Es folgte die endgültige Live-Schaltung der Replikation. Zu Beginn lief die Replikation sauber durch. Leider mussten man feststellen, dass erst mit laufender Produktion und über eine längere Zeitspanne nach und nach weitere Probleme aufgeschlagen sind, welche zusammen mit dem Support in Neuseeland Schritt für Schritt abgearbeitet werden mussten. Da in dieser Zeit natürlich der eigentliche Zweck unseres Projektes, nämlich eine Hochverfügbarkeitsumgebung bereitzustellen nicht gewährleistet werden konnte, entschlossen wir uns parallel eine Standby Datenbank mit Dbvisit Standby aufzubauen. Sowohl aus Performancegesichtspunkten als auch Lizenztechnisch war dies kein Problem. Dbvisit stellte uns die Standby-Lizenz kostenfrei zur Verfügung, solange die Probleme der Replikation nicht gelöst waren. Die Probleme bezogen sich auf CLOBs die nicht sauber auf der Replikationsseite abgearbeitet werden konnten und immer wieder zu Problemen führten.

Logisch oder doch nicht?

Alles in allem entstehen für den regelmäßigen Betrieb, einen möglichen Switchover und auch für den Failover eine Reihe von Checks und Abläufen, die geschaffen werden müssen um eine reibungslose Replikation und eine sichere Hochverfügbarkeitsumgebung zu gewährleisten. Schwierigkeiten, Komplexität und Einschränkungen führen am Ende zu einer unerwarteten Lösung. Zusammen mit Dbvisit Standby fährt man sozusagen eine Doppelstrategie und macht sich die Vorteile beider Lösungen zu nutze. Mit Dbvisit Standby erreicht man, dass alle Einschränkungen bzgl. nicht replizierter DDLs, SYS und SYSTEM Schemas sowie Datentypen umgangen werden. Mit der Replikation erreicht man, dass Verluste der eigentlichen Produktionsdaten so gering wie möglich gehalten werden. Alle verbliebenen Konfliktherde werden dabei zunächst einfach ausgeklammert und man kann sich in Ruhe ihrer Lösung in einer Testumgebung zuwenden. Ein weiterer Vorteil ergibt sich aus der einfacheren Handhabung von Dbvisit Standby im Falle eines Switchovers. Dieser gestaltet sich wesentlich unkomplizierter und eignet sich somit zum Einsatz für Wartungsarbeiten. Die Replikation wird solange einfach ausgesetzt. Bei einem Failover, also einem Totalausfall des Primary-Servers hat man nun die Möglichkeit die Standby-Datenbank zu aktivieren und im Zweifel mit einem DB Compare verloren gegangene Daten die per Replikation noch die andere Seite erreicht haben einzuspielen.

Eignet sich nun also eine logische Replikation um eine Hochverfügbarkeitsumgebung zu realisieren? Einfachste Antwort: Es kommt darauf an. Im vorliegenden Fall sehen wir, dass eine Reihe von Problemen und eine in all seinen Möglichkeiten genutzte Oracle Datenbank wenn auch in der Standard Edition die logische Replikation an ihre Grenzen bringt. Je komplexer die Datenbankstrukutr also, desto weniger empfiehlt es sich allein auf die logische Replikation zu setzen. Kommen noch Wünsche

wie ein bedienungsfreundlicher Switchover hinzu, muss man sich zwangsläufig entscheiden und sich für die einfachere Standby-Lösung mit Zeitversatz entscheiden oder letztlich Mehrkosten in Richtung Enterprise Edition und Data Guard oder anderen Lösungen in Kauf nehmen.

Kontaktadresse:

Sebastian Winkler
CarajanDB GmbH
Siemensstraße 25
D-50374 Erftstadt
Telefon: +49 (22 35) – 1 70 91 88
Fax: +49 (22 35) – 1 70 89 79
Mobil: +49 (1 75) – 8 64 90 61
E-Mail sebastian.winkler@carajandb.com
Internet: www.carajandb.com