Datenbank-Konsolidierung mit Multitenant und In Memory

Jens-Christian Pokolm, Postbank Systems AG

Bei der Konsolidierung von Datenbanken auf Servern kommt es immer wieder zum Problem mit der Menge an Real-Time-Prozessen und einer steigender Anzahl an Instanzen. Mit Nutzung der In-Memory-Option wächst der Wunsch, die im Verhältnis zu Festplatten hochpreisige Ressource „Hauptspeicher" möglichst permanent auszunutzen. Wenn beide Anforderungen aufeinandertreffen, helfen nachfolgende Informationen ein Stück weiter.

Heute verfügbare Datenbank-Server haben oft eine hohe Anzahl an Prozessoren. Das Unternehmen des Autors setzt bei der Datenbank-Konsolidierung in der Regel Systeme mit 16 oder 32 Cores für ihre Belange ein – grundsätzlich als Stretched-RAC. Jede Datenbank, das darunterliegende ASM sowie die Clusterware (und gegebenenfalls weitere Überwachungsmonitore und Tools) benötigen auf den Servern dazu Real-Time-Prozesse *(siehe Abbildung 1)*.

Auf extrem vielen Prozessoren stehen inzwischen HyperThreads zur Verfügung und präsentieren sich im Betriebssystem und teilweise auch in der Datenbank als angeblich vollständige CPUs. Dies ist aus Sicht der Real-Time-Prozesse jedoch nicht korrekt. Letztendlich stehen ausschließlich die physischen Cores für diese Prozessklasse zur Verfügung. Auf der Datenbank-Ebene sprechen wir an dieser Stelle primär über die LMS-Prozesse auf dem System. Als Daumenregel sollten auf einem Server niemals mehr Real-Time-Prozesse als physische Cores ihren Dienst verrichten. Damit war der massiven Konsolidierung oftmals ein frühes Ende beschert – obwohl eigentlich zumeist noch ausreichend freie Prozessorleistung zur Verfügung stand *(siehe auch MOS-Notes 558185.1 und 1392248.1)*.

Multitenant

Die Einführung der Multitenant-Option (Mandantenfähigkeit) – zumeist einfach „Pluggable-Database" genannten – hat diese Grenzen ein stückweit verschoben. Deren Konstrukt besteht darin, systeminterne Objekte sowie Prozesse wie das Data Dictionary, die PL-SQL-Objekte und die Hintergrund-Prozesse in einer globalen (geteilten) Umgebung zur Verfügung zu stellen – ähnlich einem HyperVisor bei virtuellen Servern. Diese Umgebung nennt Oracle den „Root-Container" (CDB) und die darunter laufenden Datenbanken dann „Pluggable Database" (PDB).

In den Datafiles der PDB werden nur noch die Deltas zur CDB gespeichert – dies gilt für die PDB-eigenen vollständigen Objekte sowie auch für die zusätzlichen Daten im Data Dictionary. Aus Sicht der Konsolidierung eine sehr interessante Funktion – denn nun können sich mehrere PDBs einen Großteil der initialen Datenbestände einer Datenbank quasi teilen. Dies gilt neben den klassischen Tablespaces (TEMP / SYSTEM / SYSAUX / UNDO etc.) auch für die SGA und die Basis-Prozesse der Datenbank. Dadurch sind die Konsolidierungssysteme sowohl im Storage als auch bei den Real-Time-Prozessen entlastet, denn nun teilen sich mehrere Instanzen auf einem Knoten einen LMS.

In dem Konstrukt gibt es gute Möglichkeiten, um die Ressourcen von PDBs gegeneinander zu schützen – die Datenobjekte jeder PDB sind immer vollumfänglich gegeneinander gesichert. Vorhandene Ressourcen lassen sich weit besser managen als dies bei eigenständigen (autarken) Datenbanken der Fall war – denn hier bestand nur die Option mittels „CPU_COUNT" einen Maximalwert zu definieren. Sobald ein CPU-Over-Provisioning-Ansatz gefahren wurde, war das Ganze dann ein stückweit Makulatur.

Mittels Ressourcen-Manager können den PDBs nun untereinander garantierte sowie maximal mögliche Ressourcen definiert werden. Der zugrunde liegenden CDB wird wie bisher mittels „CPU_COUNT"-Limitierungen nach oben gesetzt. Die feingranulare Abstufung von Benutzern innerhalb einer PDB ist weiterhin über Consumer-Groups möglich.

In Memory

Die mit der Version 12.1.0.2 eingeführte In-Memory-Option ist abweichend von den Buffer-Konzepten kein „Pool" sondern ein In-Memory-Store. Beim Pool werden durch LRU-Mechanismen Objekte auch wieder verdrängt, sofern diese nicht mehr genutzt werden oder der Platz im Speicher nicht mehr ausreicht. Die In-Memory-Option hingegen gibt einmal belegten Speicher nur auf explizite Anweisung wieder frei. Im Auslieferungszustand ist die Option (endlich einmal) deaktiviert *(siehe Listing 1)*.

Zum Einschalten der Option ist mittels „SQL> alter system set inmemory_size=16G scope spfile sid='*';" die „inmemory_size" zu setzen. Danach muss die Datenbank neu gestartet werden, um die Option zu aktivieren. Zur Kontrolle kann man die Werte dann natürlich auch in der „v$sga"-View finden. Zur Ermittlung, welche Objekte im Speicher residieren ist, die Abfrage „select owner,segment_name,populate_status from v$im_segments;" nützlich.

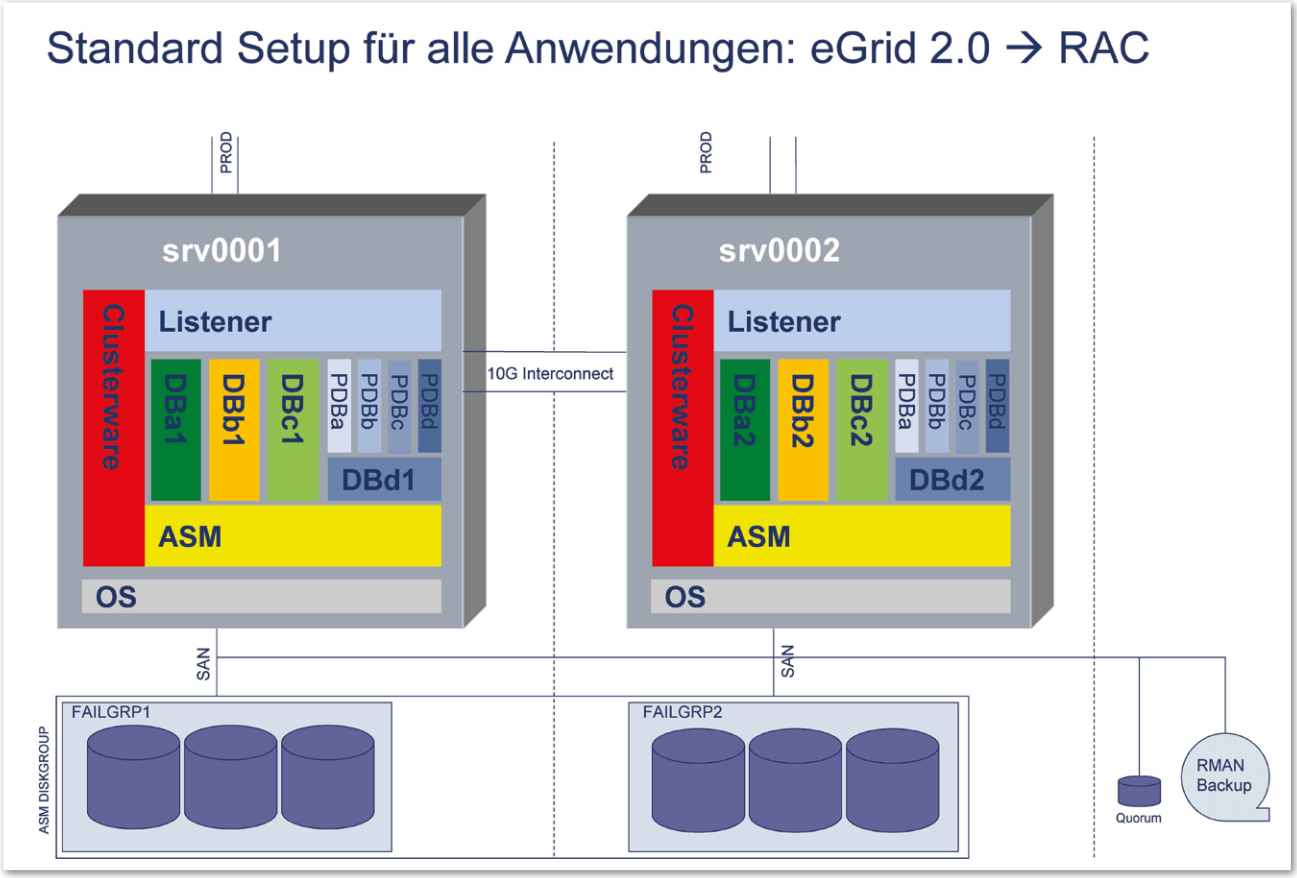

Abbildung 1: Konsolidierungsumgebung – RAC-Streched über 20 km Distanz

Wie beschrieben ist das Oracle-In-Memory Konzept ein „Store" – es muss also gezielt ausgewählt werden, welche Objekte in den Speicher kommen. Hierbei ist es interessant, dass Oracle einen „Tiering"-Ansatz unterstützt. Eine Tabelle muss daher nicht zwingend komplett in den Hauptspeicher passen. Fehlende Elemente werden bei Bedarf „on thy fly" vom Storage nachgeladen.

Es ist sehr gut, dass die Datenbestände beim Laden in den Hauptspeicher komprimiert werden. In Tests hat der Autor mit Echtdaten in der Regel einen Kompressionsfaktor zwischen 2 und 10 (median bei 5) feststellen können. Dabei ist keine „Advanced Compression"-Option erforderlich – der interne Kompressions-Algorithmus ist völlig unterschiedlich.

Eine Datenbank-Tabelle wird im System mit der Syntax „ALTER TABLE TGRP$_IPOS_AZV_$00001 INMEMORY MEMCOMPRESS FOR QUERY NO INMEMORY (COL9_PB15_ANME, COL10_PB15_ENME);" als „In Memory" markiert. Im Beispiel wurden für Abfragen uninteressante Spalten einfach ausgeklammert, sie belegen damit auch keinen kostbaren Hauptspeicher. Ohne weitere Angaben werden die so markierten Datenbestände erst beim ersten Zugriff in den Speicher kopiert. Wenn dies unmittelbar erfolgen soll, so ist das Statement um den Parameter „PRIORITY LOW|MED|HIGH|CRITICAL" zu ergänzen. In dem Fall lädt die Datenbank beim Start die Tabellen in der Reihenfolge der gesetzten Priorität in den Hauptspeicher.

Soll die In-Memory-Option im RAC genutzt werden, sind zuvor einige Parameter zwingend zu beachten, um keine negativen Nebeneffekte zu produzieren. „PARALLEL_FORCE_LOCAL" muss auf „FALSE" (default) stehen, denn leider steht nur auf den Engineered-Systems die Option zur Duplizierung des In-Memory-Stores zur Verfügung; in sonstigen Umgebungen werden die Daten in den Caches verteilt abgelegt. Anderenfalls würde der Remote-Knoten die Daten von der Disk lesen und damit die Performance massiv einbrechen. „PARALLEL_DEGREE_POLICY" muss auf „AUTO" oder „ADAPTIVE" stehen (zur Not auch auf „LIMITED" – wegen der Funktion „In-memory-parallel execution", default = „MANUAL") und „PARALLEL_DEGREE_LIMIT" dementsprechend auf einen Wert größer/gleich „2" gesetzt sein.

Oracle setzt den Parameter „INMEMORY_MAX_POPULATE_SERVERS" (Server, die den In-Memory-Store beladen) auf den Wert gleich der Anzahl CPUs. Dies ist bei Systemen mit sehr vielen Prozessoren oftmals etwas zu gut gemeint und sollte je nach Leistungsfähigkeit des Storage unter Umständen angepasst werden.

Der Nutzen

Das Lastverhalten von Datenbanken wird bei der Konsolidierung gerne zunutzen gemacht – bei klassischen Umgebungen packt der Autor Datenbanken mit unterschiedlichen Lastprofilen (Tages-/Nachtverarbeitung) sehr gerne zusammen. Dabei profitieren beide Umgebungen von den jeweils wechselseitig ungenutzten CPU/IO-Kapazitäten der anderen Datenbank.

Um ein vergleichbares Verhalten auch bei der Nutzung des In-Memory-Stores zu haben, werden Tabellen, die beispiels-

```
SQL> show parameter inmemory
NAME                                 TYPE             VALUE
------------------------------------ ---------- -------------
inmemory_clause_default              string
inmemory_force                       string           DEFAULT
inmemory_max_populate_servers        integer          4
inmemory_query                       string           ENABLE
inmemory_size                        big integer      0
inmemory_trickle_repopulate_servers_ integer          1
percent
optimizer_inmemory_aware             boolean          TRUE
```

Listing 1

weise in der Nacht von einer PDB aus dem Tagesbetrieb nicht mehr erforderlich sind, mittels „ALTER TABLE TGRP$_IPOS_AZV_$00001 NOINMEMORY;“ aus dem Speicher entfernt. Damit steht der Hauptspeicher nun für andere Objekte (PDBs) wieder zur Verfügung und kann von diesen genutzt werden, ohne dass die Funktion grundsätzlich beschränkt ist. Die für die nachtaktive Datenbank benötigten Objekte werden mit dem Parameter „INMEMORY“ markiert und wahlweise mittels SQL-Statement oder „PRIORITY“-Parameter in den RAM kopiert.

Leider bringt dieser Ansatz bei klassischen Non-PDBs nichts. Auch wenn man hier Objekte aus dem In-Memory-Store entlädt, wird die SGA deswegen nicht verkleinert. Da PDBs sich die SGA teilen, kann man dies dort zum Vorteil nutzen. Das Verfahren lässt sich über den Datenbank-Scheduler gut automatisieren und die vorherigen Zustände in entsprechenden eigenen Steuertabellen speichern. Somit ist das Verfahren auch dynamisch in der Nutzung und bedarf keiner separaten aufwändigen manuellen Pflege.

Leider ist zum aktuellen Zeitpunkt die Aufrüstung von Hauptspeicher bei steigender Modulgröße nicht ansatzweise linear. In den kommenden Jahren wird RAM jedoch Schritt für Schritt deutlich günstiger werden – parallel aber auch die Anforderungen der Anwendungen und die verarbeiteten Datenmengen massiv steigen. Daher ist davon auszugehen, dieser kleine Trick wird länger überleben – sofern denn Datenbanken mit unterschiedlichen Lastprofilen vorhanden sind. In Tests zeigten die PDBs keine messbaren Performance-Unterschiede zu den klassischen Datenbanken, daher sind sie auch mit In-Memory ein hervorragendes Mittel zur Konsolidierung.

Jens-Christian Pokolm
jens-christian.pokolm@postbank.de

Richtigstellung zum Artikel von Andrew Lacy in der letzten Ausgabe

Nachdem mein Artikel „Praxisbericht: Downgrade Datenbank Enterprise Edition auf Standard Edition One“ in der DOAG News 06/2014 erschienen ist, wurden wir darauf angesprochen, dass der letzte Satz des Artikels den Eindruck vermittelt, Kunden könnten aufgrund einer Datenbank-Migration durch einen Oracle-Platinum-Partner ein Audit vermeiden. Im besprochenen Fall hat Oracle tatsächlich das Audit vereinfacht, da wir in Abstimmung bestätigen konnten, dass durch die Migration und die damit verbundene Neulizenzierung nun korrekt lizenziert sei.

Generell ist es natürlich nicht so, dass aufgrund der Durchführung der Migration durch einen Platinum-Partner Audits vermieden werden können. Das Recht und die Initiative zu einem Lizenz-Audit liegt ausschließlich bei Oracle und wird durch diese wahrgenommen. Ich bitte um Entschuldigung, sollte dieser Satz missverstanden worden sein.

Andrew Lacy
OPITZ CONSULTING Deutschland GmbH