für den Partial Index im Vergleich zu einem vollständig aufgebauten Index auf ein konstant niedrigeres Niveau.

Schrittweise Einführung eines neuen Index

Bei sehr umfangreichen Fakten-Tabellen kann der vollständige Aufbau eines neuen Index sehr viel Zeit in Anspruch nehmen. Oft steht für die Erstellung eines neuen Index jedoch nur ein begrenztes Zeitfenster zwischen der Ausführung der ETL-Prozesse und dem Zugriff der DWH-Benutzer zur Verfügung, sodass die zur Verfügung stehende Zeit nicht für den vollständigen Aufbau des Index ausreicht.

Ist der neue Index hingegen als Partial Index erzeugt und definiert man zunächst sämtliche Tabellen-Partitionen mit „INDEXING OFF“, lässt sich der Index anschließend partitionsweise aufbauen. Dazu sind die Tabellen-Partitionen schrittweise auf „INDEXING ON“ zu setzen. Somit kann die Einführung des neuen Index auf mehrere Zeitfenster verteilt erfolgen, sodass die begrenzt zur Verfügung stehende Zeit keinen Engpass mehr darstellt. Bei diesem Verfahren sollte man abschließend den Partial Index mit „ALTER INDEX <index_name> INDEXING FULL“ in einen vollständigen Index umdefinieren, damit der Status der Index-Partitionen bei der Einführung weiterer Partial Indices unberührt bleibt.

Schrittweiser Rebuild eines Index

Soll ein bereits existierender vollständiger Index schrittweise aufgebaut werden, kann das zuvor beschriebene Verfahren für die Einführung neuer Indizes ebenfalls angewendet werden. Für den Index „BX_VERKAUF_FILIALE“ aus dem hier verwendeten Beispiel ist dabei folgender Ablauf vorzusehen:

1. Umdefinieren des vollständigen Index in einen Partial Index mit „ALTER INDEX BX_VERKAUF_FILIALE INDEXING PARTIAL“
2. Alle Tabellenpartitionen auf „INDEXING OFF“ setzen
3. Schrittweise alle Tabellen-Partitionen wieder auf „INDEXING ON“ setzen. Dabei findet automatisch der Rebuild der jeweiligen Partitionen des Index „BX_VERKAUF_FILIALE“ statt.
4. Umdefinieren des Partial Index in einen vollständigen Index mit „ALTER INDEX BX_VERKAUF_FILIALE INDEXING FULL“

Mit diesem Verfahren kann der schrittweise Rebuild auch für mehrere Partial Indices gleichzeitig erfolgen.

Fazit

Oracle Database 12*c* bietet mit der Einführung der Partial Indices mehr Flexibilität beim Aufbau der Indizes. Ihr Einsatz eignet sich besonders für große partitionierte Fakten-Tabellen im Data Warehouse. Sie ermöglichen für die Erstellung neuer und den Rebuild existierender Indizes den schrittweisen Aufbau, sodass auch kürzere Wartungszeitfenster für die Durchführung dieser Aufgaben genutzt werden können. Für Fakten-Tabellen mit besonders umfangreichen historischen Datenmengen können Partial Indices zudem genutzt werden, um den Speicherbedarf zu begrenzen.

Reinhard Mense
reinhard.mense@areto-consulting.de

Replikation großer Datenmengen in einem OLTP-System

Uwe Simon, T-Systems Telekom IT

Bei der Replikation von Daten bietet Oracle mit Read-only-Snapshots, Multimaster-Replikation, Streams oder GoldenGate die passenden Replikationsverfahren für verschiedene Anwendungsfälle an. Eine Replikation mit Read-only-Snapshots ist sehr einfach zu implementieren und für sehr viele Anwendungsfälle gut geeignet. Doch wo sind die Grenzen und wo muss man bei Problemen suchen? Am Beispiel der Replikation in einem großen OLTP-System wird gezeigt, wo diese Grenzen liegen und wie man Probleme identifizieren und lösen kann.

Die Installation einer Replikation mit Read-only-Snapshots, auch „Materialized View“ (MView) genannt, ist recht einfach und in den Handbüchern gut beschrieben. Sie bietet sich an, wenn man Daten aus einem produktiven System für reine Lesezugriffe auf einem oder mehreren anderen Systemen zur Verfügung stellen muss. Nachfolgend

```
CREATE MATERIALIZED VIEW LOG ON xxx$ta_xxx WITH PRIMARY KEY;
```

Listing 1

```
CREATE MATERIALIZED VIEW xxx$mv_xxx REFRESH FAST AS
SELECT * FROM xxx$ta_xxx@DBLINK
```

Listing 2

```
CREATE TABLE xxx$mv_yyy (id NUMBER NOT NULL, …);
CREATE MATERIALIZED VIEW xxx$mv_yyy ON PREBUILT TABLE
REFRESH FAST AS SELECT id, … FROM xxx$ta_yyy@DBLINK
```

Listing 3

werden nur die „FAST REFRESHABLE MVIEWS“ betrachtet, da sie der einzige Weg zum zeitnahen Replizieren großer Tabellen sind. Die Installation besteht je Tabelle aus zwei Schritten. Zu Beginn wird dabei auf der MView-Site noch zusätzlich ein Database-Link benötigt. Die Installation kurz zusammengefasst:

- Auf der Master-Site den MView-Log anlegen, am besten immer mit „WITH PRIMARY KEY“ (siehe Listing 1)
- Auf der MView-Site die MView anlegen (siehe Listing 2)
- Es ist geschickter, die MViews als Prebuilt Tables anzulegen. Damit kann man bei Datenbank-Kopien die MViews löschen und neu anlegen, ohne dabei die Daten wieder neu laden zu müssen. Zudem kann man dann einfach zusätzliche Spalten anlegen, die über Trigger gefüllt werden, etwa eine Spalte mit dem Zeitpunkt der letzten Replikation (siehe Listing 3)
- Jetzt fehlen nur noch die Jobs, die die MViews konsistent in einer oder mehreren Replikationsgruppen aktualisieren. Der Zuschnitt der Replikationsgruppen und deren Refresh-Intervalle sind typischerweise fachlich getrieben. Dies kann entweder über „DBMS_REPCAT.CREATE_MVIEW_REP_GROUP“ und „DBMS_REFRESH.MAKE“ erfolgen oder, falls die Replikation in Applikationslogik eingebettet werden soll, innerhalb dieser mit „DBMS_MVIEW.REFRESH (…)“

Hiermit hat man dann sehr schnell eine lauffähige Replikation implementiert, die normalerweise auch über lange Zeit recht problemlos funktioniert. Trotzdem ist es immer gut, wenn man auf Probleme vorbereitet ist. Dazu ein paar technische Daten zur betriebenen Replikationsumgebung:

- Eine Snapshot-Datenbank mit optimierter Partitionierung/Indizierung
- 231 Fast-Refreshable Materialized Views in 15 Replikationsgruppen
- Drei Tabellen werden über Oracle-Streams aktualisiert
- Sechs Datenbanken, die Massendaten aus Snapshot-Datenbanken abziehen
- Mehrere Systeme, die die neu replizierten Daten lesen
- MViews mit bis zu 256 Partitionen/Subpartitionen
- Die größte MView hat 140 GB, alle in Summe 540 GB
- Das Replikationsvolumen beträgt 100 bis 500 MLOG-Entries/Sek.
- Eine Replikationsgruppe enthält 50 Prozent der Änderungen
- Besonderheit: Zeitstempel auf MView-Site zur Identifizierung neu replizierter Datensätze

Wenn die Replikation nicht gut läuft

Nachfolgend sind SQL-Statements für die Abfragen aufgelistet. Die meisten Informationen erhält man zwar auch per Mausklick in Grid Control, die SQL-Statements sind aber beispielsweise für die Integration in einem externen Monitoring-Tool. Dass die Replikation nicht wie gewünscht läuft, erkennt man typischerweise daran, dass die Daten auf der MView-Site zu alt sind, also der letzte abgeschlossene Refresh zu lange her ist. Dies zeigt ein Statement (siehe Listing 4). Als Nächstes stellt sich die Frage: „Läuft die Replikation schon/noch oder liefert sie Fehler?“ (siehe Listing 5).

Bei der Replikation ist es immer gut, wenn man die historischen Laufzeiten der Replikationszyklen kennt. Einerseits sieht man dann schnell, dass es

```
SELECT owner, mview_name, master_link, build_mode, contai-
ner_name, last_refresh_type, last_refresh_date
FROM dba_mviews;
```

Listing 4

```
SELECT rowner, rname, j.job, j.failures, j.broken, j.last_
date,        j.this_date, j.next_date
FROM dba_refresh r JOIN dba_jobs j ON(r.job=j.job);
```

Listing 5

```
SELECT job_name, status, actual_start_date, run_duration
FROM dba_scheduler_job_run_details WHERE job_name = 'REPL_JOB'
```

Listing 6

vielleicht ausnahmsweise mal schlechter läuft, andererseits fällt auch frühzeitig auf, wenn sich – etwa bedingt durch steigendes Änderungsvolumen – die Replikationsdauer langsam verlängert. Die Abfrage in Listing 6 zeigt die Laufzeiten in dem Fall, dass die Replikationsjobs mit dem DBMS_SCHEDULER laufen. Falls sie über DBA_Jobs erfolgen, sollte diese Information auf jeden Fall durch entsprechende PL/SQL-Logik in einer eigenen Log-Tabelle gespeichert werden (siehe Abbildung 1).

Wenn die Replikation plötzlich deutlich länger läuft beziehungsweise gar nicht fertig wird, stellt sich die Frage: „Warum dauert es denn gerade jetzt so lange?“ Dafür gibt es zwei Hauptgründe:

- *Fachlich*
  Es werden deutlich mehr Daten angelegt/geändert/gelöscht und damit repliziert
- *Technisch*
  Die Replikation dauert je Änderung länger

Die Menge der geänderten/angelegten/gelöschten Datensätze kann man bei der MView-Replikation leider nicht direkt erkennen, da es dafür keine Views beziehungsweise Tabellen im Systemkatalog gibt. Mit etwas PL/SQL-Kenntnissen kann man dies zumindest abschätzen. In einem eigenen Replikations-Job wird vor und nach dem eigentlichen „DBMS_MVIEW.REFRESH(...)“ jeweils eine Abfrage gemacht (siehe Listing 7) und die Differenz dieser Werte berechnet, die man am besten zusammen mit einem Zeitstempel in eine Log-Tabelle schreibt.

```
SELECT sid, name, value
FROM v$mystat JOIN v$statname USING (statistic#)
WHERE name = ‚execute count‘;
```

Listing 7

```
DBMS_APPLICATION_INFO.SET_MODULE(
      module_name=>’REPL’,
      action_name=>’GRUPPE_1’);
```

Listing 8

```
SELECT module, action, executions, sql_text
FROM v$sqlarea WHERE module = ‘REPL’ AND action = ‘GRUPPE_1’
```

Listing 9

```
SELECT dmltype$$, COUNT(DISTINCT m_row$$) anzahl
FROM <owner>.mlog$_<tablename>
GROUP BY dmltype$$;
```

Listing 10

Wer es genauer haben will, kann die drei SQL-Statements (INSERT/UPDATE/DELETE) auf der MView-Site aus „V$SQLAREA“ ermitteln. Das geht am einfachsten, wenn man vor dem eigentlichen Refresh eine Abfrage macht (siehe Listing 8), die nach dem Refresh wieder zurückgesetzt wird. Die hier übergebenen Werte sind dann in „V$SQLAREA“ in den Spalten „MODULE“ und „ACTION“ zu finden und können dort leicht abgefragt werden (siehe Listing 9). Dabei ist dann die Differenz über die „executions“ zu bilden. Damit kann man dann das Volumen der Änderungen je Replikations-Zyklus abschätzen und später auswerten beziehungsweise vergleichen. Über diesen Weg findet man

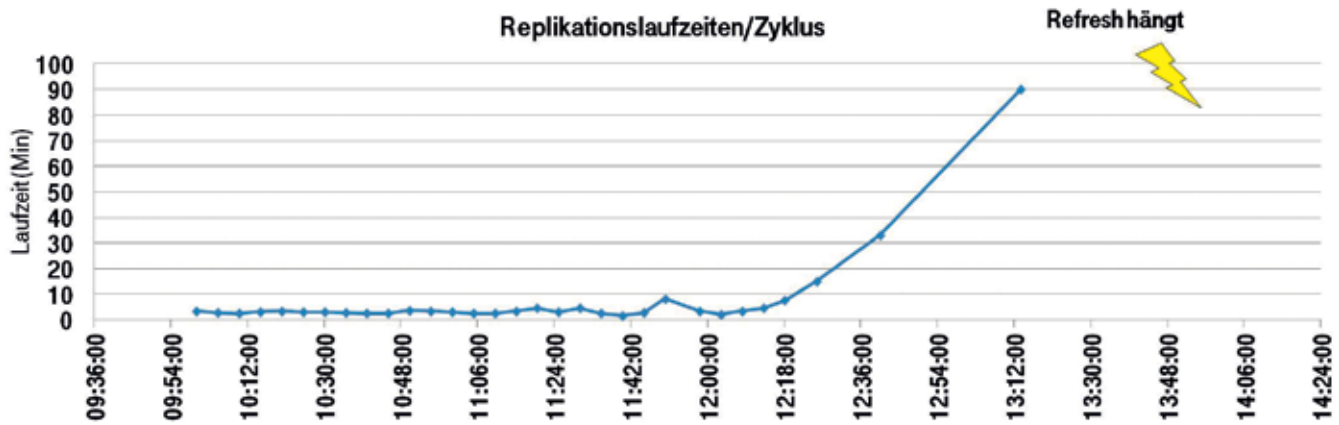

Abbildung 1: Die Replikationsdauer

	Master-Site	MView-Site
Je Replikationsgruppe		DBMS_REFRESH.REFRESH()
	DBMS_SNAPSHOT_UTL.SET_UP() UPDATE MLOG$_.... COMMIT;	
Je MView	SELECT-Zeilen für INSERT/UPDATE	Für jede Zeile UPDATE, wenn nicht vorhanden, INSERT
	SELECT der Zeilen für DELETE	Für jede Zeile DELETE
Je Replikationsgruppe	DBMS_SNAPSHOT_UTL.WRAP_UP UPDATE MLOG$_... DELETE MLOG$_... COMMIT;	

Tabelle 1

Ursache	Abhilfe
Sehr hohe I/O-Zeiten auf MView-Site	Buffer-Cache vergrößern (hilft nur, wenn Blöcke mehr als einmal gelesen werden) I/O beschleunigen (Striping hilft nur begrenzt, da nur ein Prozess schreibt/liest)
Replikation verursacht viel I/O auf der Master-Site	Refresh-Intervall zu groß Buffer-Cache zu klein
Replikation verursacht viel UNDO auf der Master-Site (Consistent Read)	Replikations-Volumen zu groß Refresh-Intervall zu groß Sehr viele Datenänderungen während eines Replikationszyklus
Fachlich	Zu viele Daten werden repliziert (Zeilen und oder Spalten)

Tabelle 2

in „V$SQLAREA“ auch leicht die Statements, die für die meiste Laufzeit verantwortlich sind.

Wie viele Daten aktuell repliziert werden müssen, kann man auf der Master-Site über ein Statement erkennen (siehe Listing 10).

Wenn man in den Tabellen auf der Master- und der MView-Site jeweils eine Zeitstempel-Spalte hat, in der auf der Master-Site die Zeit der letzten Änderung (CHANGED_AT) und auf der MView-Site die Zeit der Replikation (REPLICATED_AT) gespeichert werden, kann man relativ einfach die zeitliche Aktualität der MView-Site auf Datensatzbasis ermitteln. Damit dies auf der MView-Site möglich ist, muss man die MViews mittels „PREBUILT“-Table installieren. Dann kann diese zusätzliche Spalte durch einen „INSERT-/UPDATE“-Trigger mit dem „SYSDATE“ gefüllt werden. Wenn das fachliche Volumen keine ausreichende Erklärung für die Replikationsdauer ist, lohnt es sich, die internen Details der Replikation zu betrachten.

Wo die Replikation ihre Zeit verbringt

Um zu verstehen, wo die Replikation Zeit verbraucht, schauen wir zuerst auf den schematischen Ablauf. Alle Aktionen werden beim Refresh dabei von der MView-Site aus angestoßen (siehe Tabelle 1). Nachdem die einzelnen Schritte während eines Replikationszyklus identifiziert sind, stellt sich natürlich die Frage nach deren Laufzeiten.

Im Beispiel von Abbildung 2 fällt auf, dass die Replikation auf der MView-Site hauptsächlich auf I/O wartet. Dies ist auch leicht erklärbar. Auf der Master-Site wurden die Daten vieler Prozesse während der Verarbeitung in den Buffer-Cache geladen und sie bleiben dann auch einige Zeit dort. Daher müssen sie zur Replikation nicht mehr physikalisch gelesen werden. Auf der MView-Site wird jedoch auf diese Daten das erste Mal während der Replikation zugegriffen, daher gibt es hier die deutlich höheren I/O-Wartezeiten. Wenn die Replikation nicht mehr gut läuft, sieht das Bild etwas anders aus (siehe Abbildung 3).

Im schlimmsten Fall ist die Replikation auf der Master-Site so lange allein mit dem „UNDO“ beschäftigt, dass pro Sekunde weniger Datensätze durch die Replikation gelesen/verarbeitet werden, als die Applikation an Änderungen produziert. Ab diesem Zeitpunkt wird die Replikation – ohne Zeiten mit deutlich niedrigerem Änderungsvolumen – nicht mehr aufholen können.

Die Replikation beschleunigen

Wenn klar ist, wie die Laufzeit zustande kommt, stellt sich die Frage: „Wie kann man diese Zeiten reduzieren?“ (siehe Tabelle 2). Falls die aktuell laufende Replikation schon sehr lange überfällig ist, kann es sinnvoll sein, sie zu terminieren und neu zu starten. Damit kann ge-

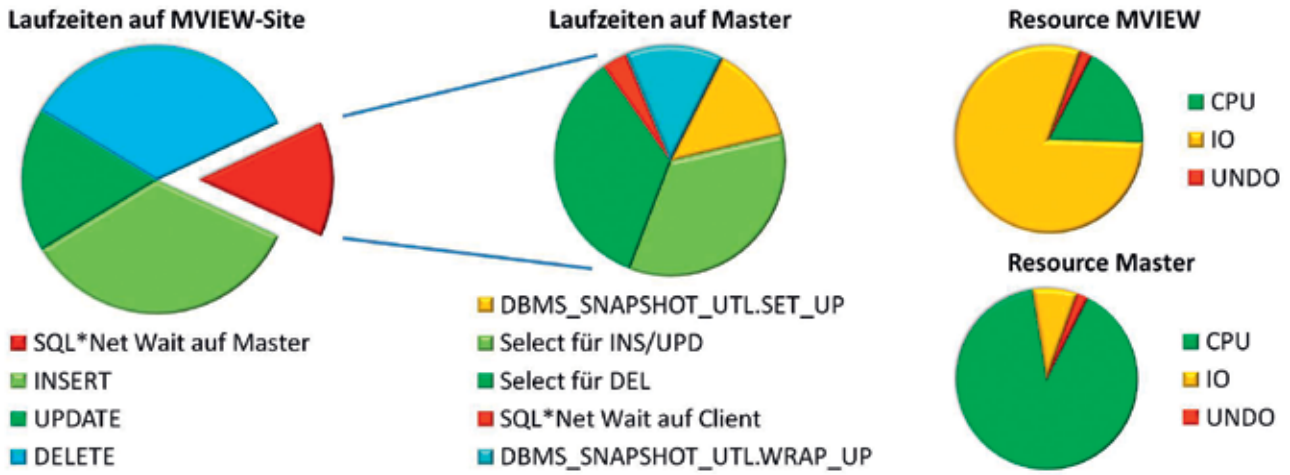

Abbildung 2: Laufzeitverteilung – Replikation läuft gut

gebenenfalls wesentlich schneller eine aktuelle MView erreicht werden. Beim Neustart ist zu beachten, dass eventuell wesentlich mehr I/O auf der Master-Site anfällt, da die zu replizierenden Daten nicht mehr im Buffer-Cache liegen.

Zur fachlichen Optimierung stehen folgende Maßnahmen zur Verfügung:

- Nur notwendige Rows/Columns replizieren
- Replikations-Reihenfolge in einer Gruppe von Tabellen, die viel UNDO erzeugen, zu Tabellen, die wenig UNDO erzeugen (reduziert gegebenenfalls Zeiten beim Zugriff auf UNDO für das konsistente Lesen)
- I/O auf MView-Site reduzieren (Buffer-Cache vergrößern)
- Nur notwendige Indizes auf MView-Site
- I/O auf MView-Site beschleunigen (etwa durch Indizes auf Solid State Disk)
- UNDO beschleunigen (UNDO-Tablespace auf Solid State Disk)
- Replikationsgruppen verkleinern
- „Steuern der Änderungsrate auf Master-Site“ in Abhängigkeit der MLOG-Größen (etwa unkritische Batch-Prozesse verlangsamen, damit das Replikations-Volumen pro Zeit immer größer ist als das Änderungs-Volumen pro Zeit)
- Auszeiten der Datenbanken mit MViews so kurz wie möglich halten beziehungsweise mit weiterlaufendem MView-Refresh, sonst werden eventuell alle anderen MViews auf der gleichen Basis-Tabelle deutlich langsamer, da der MLOG wächst
- Regelmäßiges Verkleinern der MView-Logs (nach Migrationen, Auszeiten etc.) mit „ALTER MATERIALIZED VIEW LOG ON xxx SHRINK SPACE COMPACT“. Dies sollte besonders nach einem starken Anwachsen der MLOGs erfolgen

Security-Aspekte der Replikation

Da die Replikation Datenbank-Links verwendet, gelten die gleichen Security-Anforderungen, wie sie beim sonstigen Einsatz von Datenbank-Links bestehen. Der wichtigste Aspekt dabei ist, dass der Account, mit dem auf die Master-Site zugegriffen wird, nur die unbedingt notwendigen Privilegien hat (auf jeden Fall nicht den Eigentümer der zu replizierenden Tabellen nutzen). Hierbei ist zu beachten, dass die Accounts, die auf der Master-Site für die Replikation angelegt werden, nicht nur Zugriff auf die zu replizierenden Tabel-

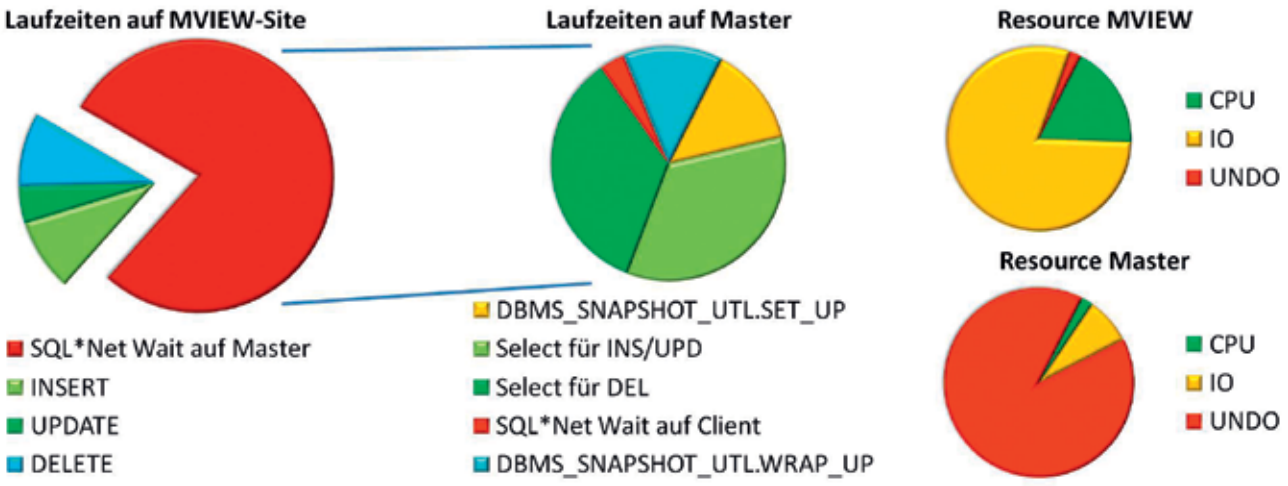

Abbildung 3: Laufzeitverteilung – Replikation läuft schlecht

Bugs zur Replikation
Mit Oracle 11.2.0.1 ORA-32329 beim Anlegen einer MView Bug 9369183: MVIEW WITH PREBUILT TABLE ON SELECT FROM REMOTE TABLE RETURNS ORA-32349 MView-Name muss unterschiedlich zum Namen der Basistabelle sein. Dafür gibt es Patch 9369183.
Master-Site 10.2.0.4, MView-Site 11.2.0.1, Dump während REFRESH Bug 5879082: Dump[kghfrf] during refresh of MView. Dafür gibt es für 10.2.0.4 den Patch 5879082.
Es werden keine Refresh mehr ausgeführt. Job-Query-Processes werden nicht gestartet Bug 10103086: wrong result for query with order by and rownum=<constant> Fixed in 11.2.0.3. Workaround Dummy-Job, der kontinuierlich läuft.
Refresh liefert ORA-4052: error occurred when looking up remote object. Auslöser ist die Information, dass Tabellen in Master-Site aus Shared-Pool geflogen sind Bug 10210507 : ORA-2019 AFTER ALTER SYSTEM FLUSH SHARED_POOL Fixed in 11.2.0.3, Workaround PUBLIC DATABASE LINK, nicht wirklich nutzbar
MViews aus 10.2.0 DBs erscheinen nicht in DBA_REGISTERED_MViews von 11.2.0.1 Bug 9249039 : MVIEW CREATED FROM 10.2 TO 11.2 MASTER IS NOT BEING REGISTERED IN SYS.SLOG$ Fixed in 11.2.0.2

Tabelle 3

len, sondern auch auf die MLOGs der Tabellen haben müssen, wenn „FAST-REFRESHABLE MVIEW“ zum Einsatz kommt. Die Replikation macht beim FAST-Refresh ein „SELECT“ mit einem „JOIN“ zwischen Tabelle und zugehörendem „MLOG“. Ferner sollte man auf jeden Fall von „PUBLIC“- Datenbank-Links mit „CONNECT TO“ Abstand nehmen.

Die Erfahrung hat ferner gezeigt, dass es auch für ein Monitoring von Vorteil ist, wenn jedes System einen eigenen Account auf der Master-Site hat. Damit kann man bei Performance-Problemen in „V$SESSION“ die Replikations-Sessions auf der Master-Site leichter unterscheiden, besonders, wenn mehrere MView-Sites auf einem Server laufen.

Performance-Probleme beim Installieren von MViews

Eine eigene Kategorie von Problemen kann beim Installieren von MViews auftreten. Einige davon findet man leider nicht so einfach auf Testsystemen. Dazu ein kleines Beispiel:

- Auf der Master-Site im laufenden Betrieb, etwa um die Auszeit zu minimieren, eine große neue Tabelle mit „INSERT AS SELECT“ anlegen
- MLOG auf der Tabelle anlegen
- Eine Prebuilt-Table auf der MView-Site anlegen
- MView anlegen
- Complete Refresh starten
- Der Complete Refresh dauert deutlich länger als im Test

Das Problem ist in diesem Fall der „Delayed Block Cleanout“, der sich besonders in Systemen mit vielen Sessions/Transaktionen negativ auswirken kann. Dieser fällt immer dann an, wenn Blöcke mit offenen Transaktionen vom DB-Writer auf Disk zurückgeschrieben werden müssen. Dann wird beim Lesen geprüft, ob der Stand auf Disk noch aktuell ist, weil ein COMMIT ausgeführt wurde, oder ob die Transaktion noch offen ist. Ist die Transaktion abgeschlossen, wird der Block Cleanout durchgeführt und über den DB-Writer auf Platte geschrieben, damit es beim nächsten Zugriff wieder schneller geht. Ob diese Block Cleanouts häufig auftreten, kann man prüfen (siehe Listing 11).

Warum werden die Delayed Block Cleanouts bei der Replikation zum Problem?

Zugriffe über Database-Links führen leider nicht dazu, dass die Blöcke nach erfolgtem Cleanout wieder zurückgeschrieben werden. Die Folge ist, dass die Cleanouts mit steigender Laufzeit des SELECT immer mehr Zeit beanspruchen – das Suchen im UNDO dauert dann immer länger – und damit das Lesen der Daten immer langsamer wird. Um den Delayed Block Cleanout während der initialen Replikation zu verhindern, kann vor dem Abzug einfach ein FULLTABLE-Scan auf der zu replizierenden Tabelle gemacht werden (siehe Listing 12). Damit kann man dann den Refresh (Größe der MView im 100-GB-Bereich) deutlich beschleunigen, obwohl vorher noch zusätzlich der FULLSCAN laufen muss. Das hat bis Oracle 10.2 gut funktioniert. Aber auch hier gibt es eine Ausnahme. Ein Select mit Parallel-Query schreibt ebenfalls den Block Cleanout nicht zurück. Hier muss man also gegebenenfalls manuell parallelisieren (etwa n Sessions parallel für n Partitionen).

Mit der Version 11*g* hat Oracle den FULLTABLE-Scan beschleunigt – was ja eigentlich eine gute Sache ist. Der Scan macht per Default die Disk-Zugriffe statt mit „SCATTERED READ“ nun mit „DIRECT READ“, was den Buffer-Cache umgeht. Dies hat jedoch den Nebeneffekt, dass wiederum kei-

```
SELECT name, value FROM v$sysstat
WHERE name LIKE ‚deferred%block cleanout%‘;
```

Listing 11

```
SELECT /*+ FULL(t) */ count(*) from tab t;
```

Listing 12

```
ALTER SESSION SET EVENTS
      ‘10949 trace name context forever, level 1’;
```

Listing 13

```
DBMS_REFRESH.REFRESH(’xxx’,’C’,atomic_refresh=>FALSE);
```

Listing 14

ne sauberen Cleanouts gemacht werden. Hierfür biete Oracle einen Workaround (siehe Listing 13).

Ein kleiner Tipp: Bei einem Complete Refresh großer Tabellen, die schon auf der MView-Site gefüllt sind, sollte man, falls fachlich möglich, eine Abfrage machen (siehe Listing 14), dann wird die MView zuerst mit TRUNCATE gelöscht, statt per Default mit „DELETE“. Das beschleunigt den Komplettaufbau der MView deutlich, führt aber dazu, dass auf der MView-Site die MView während des Refresh für alle Sessions erst mal leer ist.

Datenbank-Kopien und Replikation

Beim Benutzen einer Datenbank-Replikation muss man sehr vorsichtig beim Kopieren dieser Datenbank sein. Sowohl Datenbank-Kopien über RMAN als auch solche per Export/Import enthalten alle Replikations-Objekte, Datenbank-Links (inklusive deren Kennworte) etc. Hier muss unbedingt sichergestellt sein, dass die 1:1-Kopie nicht auf die Original-/Produktions-Datenbank zugreifen kann. Folgende Probleme führen sonst im schlimmsten Fall zu Datenverlusten in der produktiven Umgebung:

- MLOG wird durch Refresh auf der falschen Datenbank geleert
- MLOG wächst, weil die 1:1-Kopie noch für den MLOG registriert ist, aber kein Refresh mehr erfolgt

Die sicherste Lösung sind durch Firewalls abgeschottete Umgebungen, bei denen es also keine Zugriffsmöglichkeit von einer Umgebung auf eine andere (wie Test/Entwicklung auf Produktion) gibt. Falls eine Firewall nicht möglich ist, muss die Datenbank-Kopie unbedingt mit abgeschalteten „DBMS_JOBS“ und „DBMS_SCHEDULER“ hochgefahren werden; dann muss man als Erstes alle Datenbank-Links löschen und alle lokalen Kennwörter ändern (was man sowieso tun sollte). Anschließend kann die Datenbank-Kopie gefahrlos gestartet werden. Diese Vorgehensweise sollte man auch bei allen Datenbanken mit Datenbank-Links nutzen.

Bei der Datenbank-Kopie kann man sich die Arbeit deutlich vereinfachen, wenn die Materialized Views über „ON PREBULT TABLE“ angelegt wurden. Dann lassen sich die Materialized Views auch bei geänderten Datenbank-Namen ohne erneute Kopie der Daten anlegen. Die Daten bleiben auch nach dem „DROP MATERIALIZED VIEW“ in der Kopie enthalten. Damit spart man sich besonders bei großen Datenmengen viel Zeit. Dabei ist unbedingt darauf zu achten, dass alle beteiligten Datenbanken transaktionsgenau kopiert werden (wie bei allen Datenbanken, zwischen denen Datenbank-Links genutzt werden), da es andernfalls zu Daten-Inkonsistenzen kommen kann.

Bugs im Umfeld von MView-Replikation

Zu guter Letzt noch ein paar Bugs, über die man beim Einsatz der MView-Replikation (besonders bei größeren Replikations-Umgebungen) stolpern kann (siehe Tabelle 3).

Fazit

Die Read-only-Snapshot-Replikation ist auch in größeren Umgebungen sehr gut nutzbar. Mit ein paar SQL-Abfragen, die man ins Monitoring aufnehmen kann, können alle kritischen Situationen erkannt werden. Man muss aber – wie bei jedem anderen Verfahren auch – die Grenzen der Snapshot-Replikation kennen. Der wesentliche Nachteil ist die fehlende Skalierung, da die Snapshot-Replikation je Replikationsgruppe nur durch einen Prozess erfolgt. Hier muss man aufpassen, dass man die kritische Grenze nicht erreicht oder – noch schlimmer – überschreitet.

Uwe Simon
uwe.simon@t-systems.com

Kurz gemeldet: Oracle WebCenter Portfolio Updates

Die neue Version bringt das Einbinden neuer mobiler Apps, „Bring Your Own Device“-Unterstützung (BYOD) sowie umfangreiche Entwicklungswerkzeuge für mobile Portale und Webseiten.