Replikation großer Datenmengen in einem OLTP-System

Uwe Simon
T-Systems International GmbH
Bonn

Schlüsselworte
Oracle, Replikation, Materialized-Views, Readonly-Snapshots, Oracle-Streams, Performance

Einleitung
Häufig gibt es die Anforderung Daten, die in einer Datenbank aktualisiert werden, zusätzlich in einer anderen Datenbank zum Lesen bereitzustellen. Gründe hierfür sind z.B. Entkopplung der Systeme, Vermeidung von Onlinezugriffen über ein langsames WAN oder Skalierung.
Das Oracle-RDBMS bietet für unterschiedliche Anforderungen passende Replikationsmechanismen wie z.B. Readonly-Snapshot-Replication, Multimaster-Replikation, Oracle-Streams an. Für viele Aufgabenstellungen kann die Readonly-Snapshot-Replikation mit wenig Aufwand aufgesetzt werden. Diese ist dann sehr schnell funktionsfähig und auch sehr zuverlässig. Aber was passiert, wenn das replizierte Datenvolumen bzw. das Datenänderungsvolumen stark ansteigt?

Am Beispiel eines großen CRM-Systems wird dargestellt, wie die Readonly-Snapshot-Replication für eine Skalierung und Systementkopplung genutzt werden kann. Es wird auch gezeigt, wo deren Limitierungen liegen, wie man die Limitierungen lokalisieren kann und welche Maßnahmen zur Steigerung des Durchsatzes genutzt werden können.

Am Ende gibt es noch einen kurzen Ausblick, wie ein Teil der Limitierungen mit Oracle-Streams gelöst werden kann und welche Herausforderungen hierbei zu lösen sind.

Allgemeines zur Replikation
Oracle stellt eigentlich für jede Aufgabenstellung das passende Replikationsverfahren zur Verfügung. Die Verfahren lassen sich grob wie folgt kategorisieren

Replikationsverfahren	**Anwendungsfälle**
Readonly-Snapshots	Master-Site ändert ggf. mehrere Slave-Sites lesen. Replikation asynchron über DB-Job oder synchron über ON COMMIT
Updateable-Snapshots	Master-Site ändert viel, ggf. mehrere Slave-Sites haben wenige Änderungen
Multi-Master-Replikation	Alle Sites sind gleichberechtig und ändern alle
Streams-Replikation	Wie Multi-Master Änderungen werden aus Redolog extrahiert.

Tabelle.1:Replikationsverfahren

Dabei hat jedes Verfahren Vorteile, Nachteile und Limitierungen. Je nach Anforderungen muss man sich hier bewusst für ein Verfahren entscheiden.

Besonderheiten bei der Replikation im OLTP System
In einem OLTP-System können nur Replikationsverfahren genutzt werden, die „schnell“ die geänderten Daten bewegen können. Hier geht es ja darum möglichst schnell die geänderten Datensätze

von z.B. Kunden oder Verträgen auf anderen Systemen zur Verfügung zu stellen. Da sich aber – je Stunde- nur prozentual sehr wenige dieser Datensätze ändern, scheiden Verfahren wie Snapshots mit FULL-Refresh von vornherein aus.

In OLTP-Systemen wird häufig auch eine Historisierung genutzt bzw. muss auch rechtlichen Gründen genutzt werden (damit man z.B. sehen kann welchen Vertrag ein Kunde wann hatte) . Dies vergrößert die Größe der zu replizierenden Tabellen.

In einem OLTP-System werden die Daten typischerweise von vielen Usern parallel erfasst bzw. von vielen Prozessen parallel geändert. Dies führt zu vielen parallel laufenden, schreibenden Transaktionen. Dadurch kann in Spitzenzeiten ein sehr großes Datenänderungsvolumen anfallen.

Betriebene Replikationsumgebung
Im hier dargestellten Fall (Implementierung zu Oracle 8i Zeiten) war die Anforderung die Mehrzahl der nur lesenden Systeme über eine Lesekopie der Originaldaten mit Daten zu beliefern. Die betroffenen Daten machen dabei nur einen Teil der Datenbank aus und verursachen nur einen kleinen Teil des Redolog-Volumens. Die Daten müssen auf der Kopie zwar aktuell und konsistent aber nicht transaktionsaktuell (mehrere Transaktionen können zusammengefasst werden) sein. Eine zeitliche Verzögerung von wenigen Minuten bei der Datenaktualität ist hier fachlich möglich. Nach entsprechenden Tests zeigte sich, dass die Fast-Refreshable-Readonly-Snapshots diese Anforderungen beim erwarteten Daten-/Änderungsvolumen erfüllen. Da eine Vielzahl von Systemen betroffen war, wurde entschieden die Tabellen 1:1 zu replizieren, da es keine sinnvolle – über lange Zeit stabile - Optimierung gibt.

Hier ein paar Daten zur betriebenen Replikationsumgebung

Eine Snapshot-Datenbank mit optimierter Partitionierung/Indizierung
231 Fast-Refreshable Materialized Views in 15 Replikationsgruppen
3 Tabellen werden über Oracle-Streams aktualisiert
6 Datenbanken, die Massendaten aus Snapshot-DB abziehen
Mehrere System, die die neu replizierten Daten lesen
MVIEWs mit bis zu 256 Partitionen/Subpartitionen
Größte MVIEW 140GB in Summe 540GB
Replikationsvolumen 100-500 MLOG-Entries/Sek. Eine Replikationsgruppe enthält 50% der Änderungen.
Besonderheit: Zeitstempel zur Identifizierung neu replizierter Datensätze

Tabelle.2:Daten zur Umgebung

Die Replikationsumgebung besteht aus einer Master-Site, auf der die „Originaldaten“ gepflegt werden und einer Snapshot-Site von der die nachgelagerten Systeme lesen. Die Datenaktualität wird durch entsprechend kurze Refreshzyklen sichergestellt. Systeme die „transaktionsaktuelle“ Daten benötigen greifen auf die Master-Site zu.

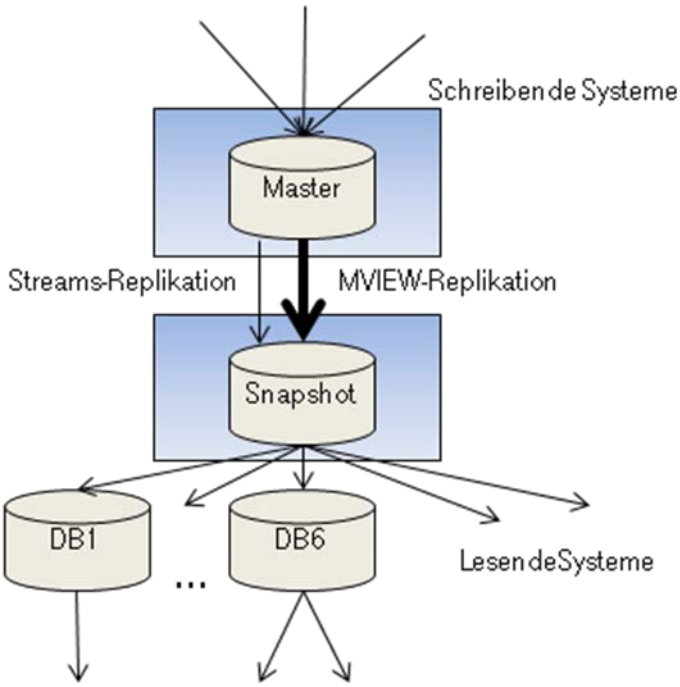

Bild 1:Schemastische Übersicht der Replikationsumgebung

Allgemeines zur Snapshot(MVIEW)-Installation
Die Installation von Readonly-Snapshots ist recht einfach und in den Handbücher auch gut beschrieben. Sie besteht je Tabelle aus 2 Schritten. Initial wird dabei auf der MVIEW-Site noch ein Database-Link benötigt.
Hier einmal die Installation als kurze Zusammenfassung.
Auf Master-Site den Materialized View Log anlegen:

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

Auf MVIEW-Site die MVIEW anlegen

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

Geschickter ist es die MVIEWs als Prebuild Tables anzulegen. Dann kann man bei DB-Kopien die MVIEW löschen ohne abei die Daten zu wieder neu laden zu müssen.

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

Die Erfahrung hat gezeigt, dass es für ein Monitoring von Vorteil ist, wenn jedes System einen eigenen Account auf der Master-Site hat. Damit kann man bei Performanzproblemen die Replikations-Sessions auf der Mastersite leichter unterscheiden – insbesonders wenn mehrere MView-Sites auf einem Server laufen – sonst sind die in V$SESSION nicht zu unterscheiden.

Wird je MVIEW-Site ein Account auf der Master-Site verwendet ist zu beachten, dass der Account nicht nur Leserechte an der zu replizierenden Tabellen hat, sondern auch an den MLOGs dieser Tabellen.

Achtung:
Public-DB-Links mit CONNECT TO IDENTIFIED BY sind ein absolutes Datenschutzrisiko und sollten tunlichst nicht genutzt werden (auch außerhalb der Replikation).

Monitoring
Auch bei der „recht simplen“ Materialized-View-Replikation gilt: „Wenn was schiefgehen kann, geht es irgendwas auch mal schief“. Deshalb führt kein Weg um ein sinnvolles Monitoring herum.

Mit entsprechenden Monitoring-Abfragen, die sich in die gängigen Monitoring-Tools integrieren lassen, kann dies aber frühzeitig erkannt werden. Etwas kniffelig ist dabei die sinnvolle Parametrisierung, da ein MVIEW-Refresh je nach Änderungsvolumen eben mehr oder weniger lange laufen kann..
Hierbei geht es um die fachlich motivierte Frage

„Wann ist eine Materialized-View fachlich nicht mehr aktuell genug?“

Monitoring auf MVIEW-Site
Wie alt die MVIEWs aktuell sind, ergibt sich mit

```
SELECT owner, mview_name, master_link, build_mode, container_name,
last_refresh_type, last_refresh_date
FROM dba_mviews;
```

Durch eine passende WHERE-Clause kann man sich dann die “zu alten” MVIEWS herausfiltern und ggf. alamieren.
Welche Replikationen-Jobs gerade laufen, erhält man mit

```
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);
```

Hier lassen sich analog mit passenden WHERE-Clauses die „zu lange laufenden” Replikationsjobs ermitteln.

In Oracle fehlt hier leider ein Monitoring über die Anzahl der replizierten Datensätze. Ein langlaufender Refresh bedeutet nicht immer sofort ein Problem, da ein Refresh-Zyklus zwangsläufig länger dauert, wenn mehr Datenänderungen repliziert werden müssen. Die Replikation macht je Row ein UPDATE bzw. ein INSERT+UPDATE. Das kann man in einem Replikations-Job ausnutzen indem man aus der Differenz von

```
select sid, name, value
from v$mystat join v$statname using (statistic#)
where name = 'execute count';
```

vor und nach dem Replizieren eine Abschätzung der geänderten Rows berechnet. Jede Row erhöht dabei dann den „execute count“ um 1 oder 2. Diesen Wert kann man dann zusammen mit der Replikationsdauer wegspeichern um es später auszuwerten.

Monitoring auf Master-Site
Auf der Master-Site stellen sich hauptsächlich 3 Fragen:

Wer hat MVIEWs auf einer Tabelle?

```
SELECT owner, name, mview_site, refresh_method
FROM dba_registered_mviews;
```

Wann war der letzte Refresh der MVIEWS auf den MVIEW-Sites?

```
SELECT r.owner, r.name, r.mview_site, b.owner master_owner,
b.master, b.mview_last_refresh_time
FROM dba_registered_mviews r
```

```
JOIN dba_base_table_mviews b ON(r.mview_id = b.mview_id);
```

Wie viele Daten stehen noch im MLOG von Tabelle xxx$ta_xxx?

```
SELECT COUNT(*) FROM mlog$_xxx$ta_xxx;
```

Automatisierung bei der Installation

Da die Replikation 1:1 erfolgt, konnte die Generierung der Skripts für die Prebuilt-Tables, MViews, MView-Logs etc. voll automatisiert werden. Auch die Skripts für das initiale Kopieren der Daten werden automatisch generiert. Hierdurch ließ sich der zusätzliche Aufwand bei Datenmodelländerungen sehr stark reduzieren und die Fehleranfälligkeit minimieren.

Performanz der Materialized-View-Replikation

Wenn man die Replikation installiert hat und sie eine Zeit lang erfolgreich läuft, dauert es meist nicht lange bis es irgendwann mal klemmt. Über das Monitoring kann dies zeitnah festgestellt werden. Die Laufzeiten eines Replikationszyklusses sehen in diesem Fall wie im folgenden Beispiel aus.

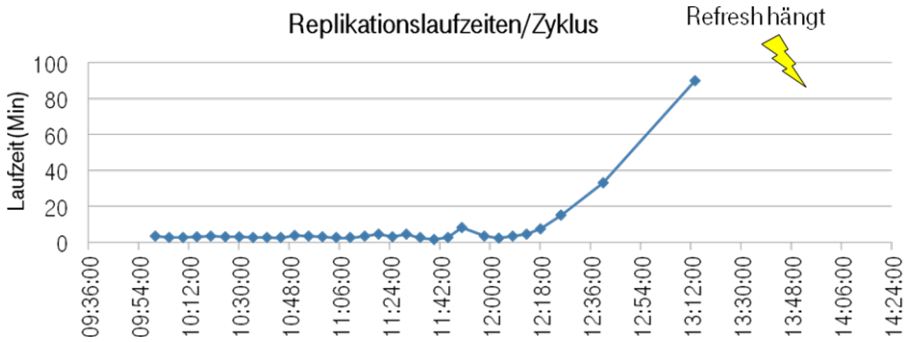

Bild 2:Replikation hat ein Problem

Nun stellt sich die Frage „Warum dauert die Replikation eigentlich so lang?“
Hierzu muss man folgende Stellen betrachten

- Größe der MLOGs auf der Mastersite
- Lesen auf der Master-Site
- Netzwerk Transfer
- Schreiben auf MVIEW-Site

Der offensichtlichste Quelle findet man meist in den MLOGs.

Größe der Tabelle	Es wurden in einem Refresh-Zyklus viele Daten geändert, dadurch hat das MLOG-Segment stark vergrößert, ist jetzt aber leer (Replikation macht hier FULLSCAN bis zur HighWatermark)
Anzahl der Rows in MLOG	Es wurden während eines Replikationszyklusses sehr viele Daten geändert. Wenn mehrere MVIEW-Sites für einen MLOG registriert sind, holt ggf. eine andere Site die Daten nicht mehr ab und verlangsamt dadurch alle anderen Sites (Rows in den MLOGs werden erst gelöscht wenn alle Sites abgeholt haben).

Tabelle.3:MLOG ist sehr groß

Wie bekommt man jetzt die „bremsende Komponente heraus“?

Um zu verstehen, warum die Replikations langsam ist und wo die Zeit verloren gehen kann, macht es Sinn sich erst mal die Zusammensetzung der Laufzeit eines Replikationszyklusses anzuschauen.
Ein Replikationszyklus besteht aus den Schritten

	Master-Site	**MVIEW-Site**
Je Refresh-group		DBMS_REFRESH.REFRESH()
	DBMS_SNAPSHOT_UTL.SET_UP() UPDATE MLOG$_.... COMMIT;	
Je MVIEW		
	SELECT Rows für INSERT/UPDATE	
		Für jede Zeile UPDATE, wenn nicht vorhanden INSERT
	SELECT der Rows für DELETE	
		Für jede Zeile DELETE
Je Refresh-group	DBMS_SNAPSHOT_UTL.WRAP_UP UPDATE MLOG$_... DELETE MLOG$_... COMMIT;	

Tabelle 4:Ablauf der Replikation

Solange es gut läuft verteilt sich die Antwortzeit wie folgt

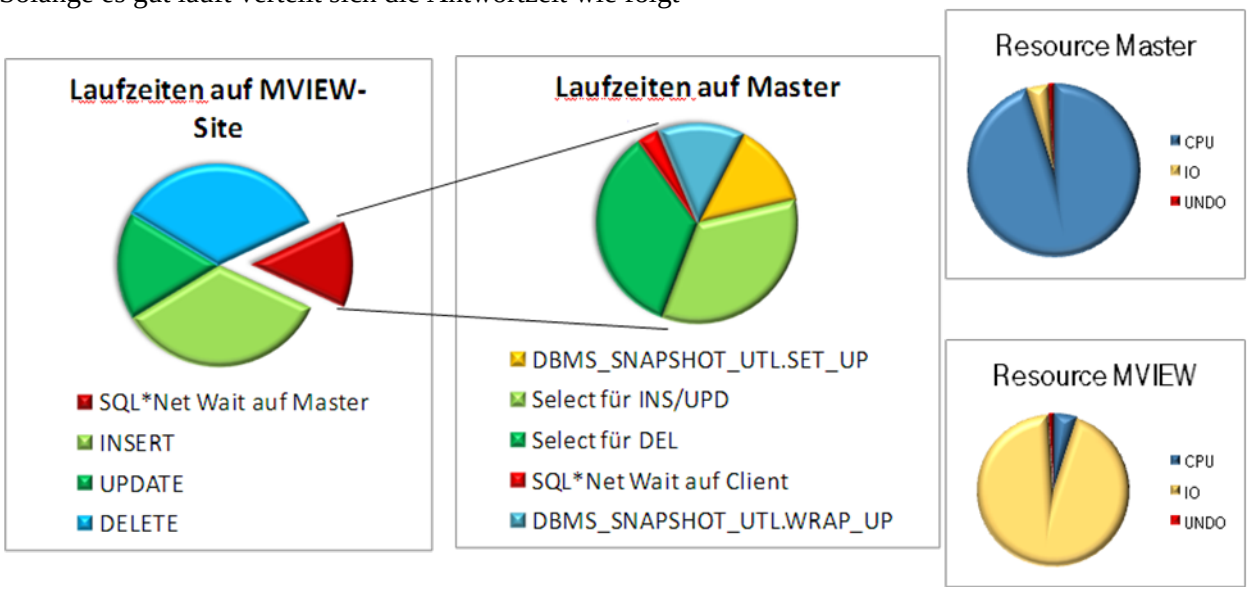

Bild 3:Ein „gut laufender“ Replikationszyklus

Wenn die Replikation „langsam“ läuft verändert sich die Verteilung oft wie folgt

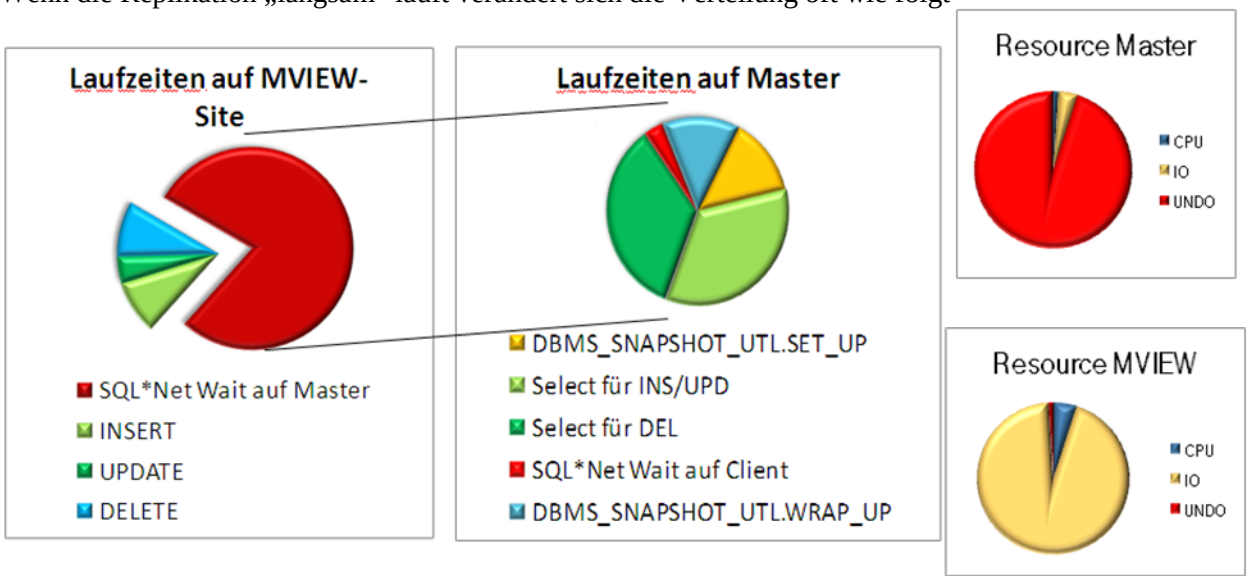

Bild 4:Ein „schlecht laufender“ Replikationszyklus

Ursachen für eine „langsame Replikation“
Wenn man nun weiß, wo die Zeit bei der Replikation verbraucht wird, kann man gezielt nach den Ursachen forschen.

Ursache	**Abhilfe**
Sehr große IO-Zeiten auf MVIEW-Site.	• Buffercache vergrößern (hilft nur, wenn Blöcke mehr als einmal gelesen werden) • IO beschleunigen (Striping hilft nur begrenzt, da nur ein Prozess schreibt/liest).
Replikation verursacht viel IO auf Master-Site	• Refreshintervall zu groß • Buffercache zu klein
Replikation verursacht viel UNDO auf Master-Site (Consistent Read)	• Replikationsvolumen zu groß • Refreshintervall zu groß • Sehr viele Datenänderungen während eines Replikationszyklus
Fachlich	• Zu viele Daten werden repliziert (Zeilen und oder Spalten)

Tabelle 5:Ursachen für“ langsame“ Replikation

Sonstiges Performanztuning der Replikation
Wenn man alle technischen Möglichkeiten ausgeschöpft hat, bleiben nur noch Änderungen an der Logik bzw. der Applikation übrig. Diese versprechen normalerweise den größten Erfolg, machen aber häufig auch den mit Abstand größten Aufwand

Nur notwendige Rows/Columns replizieren
Replikationsreihenfolge in einer Gruppe von Tabellen, die viel UNDO erzeugen, zu Tabellen, die wenig UNDO erzeugen (zögert „Umkippen“ hinaus)
IO auf MVIEW-Site reduzieren (Buffercache vergrößern).
Nur notwendige Indexe auf MVIEW-Site
IO auf MVIEW-Site beschleunigen (z.B. Indexe auf Solid-State-Disks - SSD)
UNDO-Beschleunigen (UNDO-Tablespace auf SSD)
Replikationsgruppen verkleinern
„Steuern der Änderungsrate auf Master-Site“ in Abhängigkeit der MLOG-Größen (z.B. unkritische Batchprozesse verlangsamen, damit Replikationsvolumen/Zeit immer größer als Änderungsvolumen/Zeit ist)
Auszeiten der DBs mit MVIEWs so kurz wie möglich bzw. mit weiter laufenden MVIEW-Refresh, sonst werden ggf. alle anderen MVIEWs auf der gleichen Basistabelle deutlich langsamer da MLOG wächst.
Regelmäßiges Verkleinern der MVIEW-Logs (Nach Migrationen, Auszeiten etc.) mit ALTER MATERIALIZED VIEW LOG ON xxx SHRINK SPACE COMPACT;

Tabelle 6:Fachliche Optimierungen

Performanzprobleme 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.
Ein kleines Beispiel:

1. Auf Mastersite wird im laufenden Betrieb – z.B. um die Auszeit zu minimieren - eine große neue Tabelle mit INSERT AS SELECT angelegt.
2. MLOG auf die Tabelle anlegen
3. Prebuilt Table auf MVIEW-Site anlegen
4. MVIEW Anlegen
5. Complete Refresh starten
6. Der Complete Refresh dauert deutlich länger als im Test.

Es stellt sich also die Frage: „Was klemmt hier?“

Problem ist in diesem Fall der „Delayed Block Cleanout“, der sich besonders in Systemen mit vielen Sessions/Transaktionen negativ auswirken kann. Der „Delayed Block Cleanout“ fällt immer dann an, wenn Blöcke mit offenen Transaktionen vom DB-Writer auf Disk zurückgeschrieben werden müssen. Dann muss beim Lesen geprüft werden, 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 wieder auf Platte geschrieben, damit es beim nächsten Zugriff wieder schneller geht. Ob diese Block-Cleanouts häufig auftreten, kann man mit

```
select name, value from v$sysstat
where name like 'deferred%block cleanout%';
```

prüfen.

Warum werden die Delayed Block Cleanouts bei Replikation zum Problem?

Zugriffe über Database-Links führen leider nicht dazu, dass die Blöcke nach erfolgtem Cleanout wieder zurückgeschrieben werden. Dies führt dummerweise dazu, dass die Cleanouts mit steigender Laufzeit des Selects immer mehr Zeit beanspruchen – das Suchen im UNDO dauert dann immer länger - und damit das Lesen der Daten immer langsamer wird.

Was kann man dagegen tun?
Vor einem Abzug einfach einen FULLTABLE-Scan auf der zu replizierenden Tabelle durchführen

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

Damit kann man dann den Refresh (im 100GB-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 Cleanout nicht zurück. Hier muss man also ggf. manuell parallelisieren (z.B. n Sessions parallel für n Partitionen)

Mit Oracle 11g hat Oracle den FULLTABLE-Scan beschleunigt – was ja eigentlich eine gute Sache ist. Der Scan macht nun per Default die Disk-Zugriffe statt mit SCATTERED READ nun mit DIRECT READS (umgeht den Buffercache). Das hat nun den Nebeneffekt, dass wiederum keine sauberen Cleanouts gemacht werden. Aber Oracle hat dafür aber den Workarround.

```
alter session set events '10949 trace name context forever, level 1';
```

Kleiner Tip:
Bei einem Complete-Refresh großer Tabellen sollte man, falls fachlich möglich, ein

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

machen, dann wird die MVIEW zuerst mit TRUNCATE gelöscht, statt per Default mit DELETE. Das beschleuingt den Komplettaufbau der MVIEW deutlich, führt aber dazu, dass auf der MVIEW-Site die MView für alle Sessions erst mal leer ist.

Bugs im Umfeld von MView-Replikation
Hier ein paar der Bugs, über die man beim Einsatz der MView-Replikation stolpern kann (besonders bei größeren Replikations-Umgebungen).

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. Hierfür gibt es Patch 9369183.
Mastersite 10.2.0.4, MVIEW-Site 11.2.0.1, Dump während REFRESH **Bug 5879082 Dump[kghfrf] during refresh of Mview** Hierfür gibt es für 10.2.0.4 den Patch 5879082
Es werden keine Refresh mehr ausgeführt. Job-Query-Proceses 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 das Infos über Tabellen in Mastersite 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 unter11.2.0.1 **Bug 9249039 : MVIEW CREATED FROM 10.2 TO 11.2 MASTER IS NOT BEING REGISTERED IN SYS.SLOG$** Fixed in 11.2.0.2

Datenbank-Kopien und Replikation
Wenn man eine Datenbank Replikation nutzt, muss man sehr vorsichtig beim Kopieren dieser Datenbank sein. Sowohl DB-Kopien über RMAN als auch Export/Import enthalten alle Replikationsobjekte, DB-Links (inkl. der Kennworte) etc. Hier muss unbedingt sichergestellt werden, dass die 1:1 Kopie nicht auf die Original-(Produktions-)Datenbank zugreifen kann. Die Probleme

- MLOG wird durch Refresh auf falscher DB geleert
- MLOG wächst weil 1:1 Kopie dafür registriert ist, aber keinen Refresh mehr macht

führen dabei im schlimmsten Fall zu Datenverlusten in der produktiven Umgebung..

Die sicherste Lösung sind durch Firewalls abgeschottete Umgebungen, also keine Zugriffsmöglichkeit von einer Umgebung auf eine andere (z.B. Test/Entwicklung auf Produktion).

Wenn eine Firewall nicht möglich ist, muss die Datenbankkopie unbedingt mit abgeschalteten DBMS_JOBS und DBMS_SCHEDULER hochfahren, dort alle DB-Links löschen und alle Kennworte ändern (was man sowieso tun sollte). Anschließend kann die Datenbank-Kopie gefahrlos gestartet werden.
Diese Vorgehensweise sollte man auch bei allen Datenbanken, die Database-Links enthalten, nutzen.

Bei der DB-Kopie kann man sich die Arbeit deutlich vereinfachen, wenn die Materialized-Views über ON PREBULT TABLE angelegt wurden. Dann können die Materialized-Views auch bei geänderten Datenbank-Namen ohne erneute Kopie der Daten angelegt werden (Die Daten sind ja schon in der Kopie enthalten). Damit spart man sich besonders bei großen Datenmengen viel Zeit.

Oracle-Streams
Oracle-Streams ist ein logbasiertes Replikationsverfahren, welches komplett außerhalb der produktiven Transaktionen läuft. Hierzu wird kontinuierlich mittels Logminer die Redologs analysiert, die notwendigen Daten extrahiert (Capture), zum Zielsystem transportiert (Propagate) und dort im Kontext der „original-Transaktion“ geschrieben (Apply). Hierbei wird jede betroffenes INSERT/UPDATE/DELETE auch auf der Apply-Seite ausgeführt.

Vorteile Oracle-Streams
Oracle-Streams belastet die „Ursprungstransaktionen“ nicht, es gibt keine Logtabellen, die hier gefüllt werden müssen.
Ferner bietet Oracle-Streams die Möglichkeit auf den verschiedenen Stufen zu skalieren. Wenn die Ursprungstransaktionen hierbei klein sind und hoch parallel laufen, können sie auch auf der Apply-Seite gut parallelisieren. Hier bestimmt die „langsamste“ Transaktion den Durchsatz. Damit kann eine entscheidende Limitierung der Materialized-View-Replikation aufgehoben werden.

Nachteile Oracle-Streams
Oracle-Streams analysiert immer alle Redo-Records im Redolog auch wenn nur ein kleiner Teil „zu replizierende“ Daten betreffen. Solange Capture und Propagate mit den aktuellen Daten arbeiten, haben beide einen guten Durchsatz. Dieser bricht aber dramatisch ein, wenn der Capture statt auf dem aktuellen Redolog auf einem alten Archivelog arbeiten muss. Besonders bei sehr hohem Redologaufkommen (>1GB/Min) führt dies schnell dazu, dass der Capture-Prozess nicht mehr aufholen kann. Ferner kann der RMAN dann die Archivelogfiles nicht mehr löschen, was bedingt, dass man genug Platz in der Archivelog-Destination haben muss.

Einsatz Oracle-Streams
Als Prototyp wurde eine „kleine“ Refreshgruppe der Materialized-View-Replikation auf Oracle-Streams umgestellt. Mit einem entsprechenden Skripting kann man die Oracle-Streams-Replikation recht zügig installieren. Der Aufwand war hier aber deutlich höher als beim Skripting für die Materialized-Views.

Auf der Zieldatenbank steht kein Zeitstempel zur Verfügung, über den man ermitteln kann, ob alle Daten, die auf der Quelle committed wurden, bis zu einem Zeitpunkt x repliziert sind.
Die Aktualität der Replikation wurde über einen DB-Job auf der Quelle, der regelmäßig ein UPDATE in einer einzeiligen Logtabelle mit einem Zeitstempel, gelöst. Damit ist dann auf dem Ziel leicht zu ermitteln, ob die Daten aktuell sind (auch wenn die Applikation selbst nichts geändert hat).

Probleme Oracle-Streams
Per Default ist Oracle-Streams gegenüber Materialized-View-Replication nicht besonders „fehlertolerant“. Das ist einerseits gut – man merkt wenn die Daten auseinander läuft – andererseits macht es die Fehlerbeseitigung aufwendiger. Wenn die Oracle-Streams-Replikation aus irgendwelchen Gründen „out of sync“ ist, ist es manchmal nicht trivial den Fehler zu beseitigen. Merkt der Apply-Prozess Datendivergenzen bleibt die Streams-Replikation erst mal stehen und es muss z.B. entweder zuerst der „alte Datensatz“ auf dem Ziel eingefügt werden, damit die Transaktion den UPDATE/DELETE machen kann oder es muss die Transaktion manuell durchgeführt werden und von Streams „übersprungen“ werden. Dann ist man sehr schnell in der Situation, dass der Backlog so groß wird, dass ein Aufholen nicht mehr möglich ist. Bei der Materialized-View-Replikation reicht oft einfach ein Dummy-Update auf den betroffenen Zeilen in der Quelldatenbank und die fehlenden Daten sind beim nächsten Refresh wieder auf den Zieldatenbanken.

Der Capture-Prozess scheint sehr anfällig für Probleme mit dem Shared-Pool zu sein. Mehrfach kam es dabei zu Stillständen des Capture-Prozesses. Die Bereinigung der Situation war dabei immer sehr aufwändig und endete teilweise in der Neu-Instanzierung der Oracle-Streams-Replikation.

Insgesamt hat sie sich - in unserem Umfeld - Oracle-Streams-Replikation nicht als so betriebssicher gezeigt wie es die Materialized-View-Replikation der umgestellten Refreshgruppe war. Insbesonders die Neuinstanzierung ist mit sehr großen Tabellen im laufenden Betrieb nicht machbar.

Dies war auch der Hauptgrund weshalb jetzt eine komplette Ablösung der Replikation implementiert wird.

Zusammenfassung
Die Materialized-View-Replikation ist ein einfach zu nutzendes betriebssicheres Replikationsverfahren, welches viele Szenarien abdecken kann. Kritisch wird es – wie bei allen Replikationsverfahren - wenn man an dessen Grenzen stößt (Bei MVIEWs fehlt eine Skalierung).

Oracle-Streams kann zwar gut skalieren, beim betrachteten System – welches bisher schon alles Oracle-Limits ausgelotet hat - fehlt es aber an der Zuverlässigkeit im täglichen Betrieb.

Kontaktadresse:
Uwe Simon
T-Systems International GmbH
Am Propsthof 49
D-53121 Bonn

Telefon: +49 (0) 228-181 42182
Fax: +49 (0) 228-181 42172
E-Mail uwe.simon@t-systems.com
Internet: www.t-systems.com