Replikation großer Datenmengen in einer OLTP Umgebung

Installation, Betrieb, Performanz und Tips.

uwe.simon@t-systems.com

T · · Systems

Übersicht

- Allgemeines zur Replikation
- Materialized Views (Snapshots)
  - Theorie
  - Praxis (Installation, Monitoring, Testsysteme, Performanz, Security)
  - Replikationsrelevante Oracle Bugs
- Oracle-Streams
  - Theorie
  - Praxis

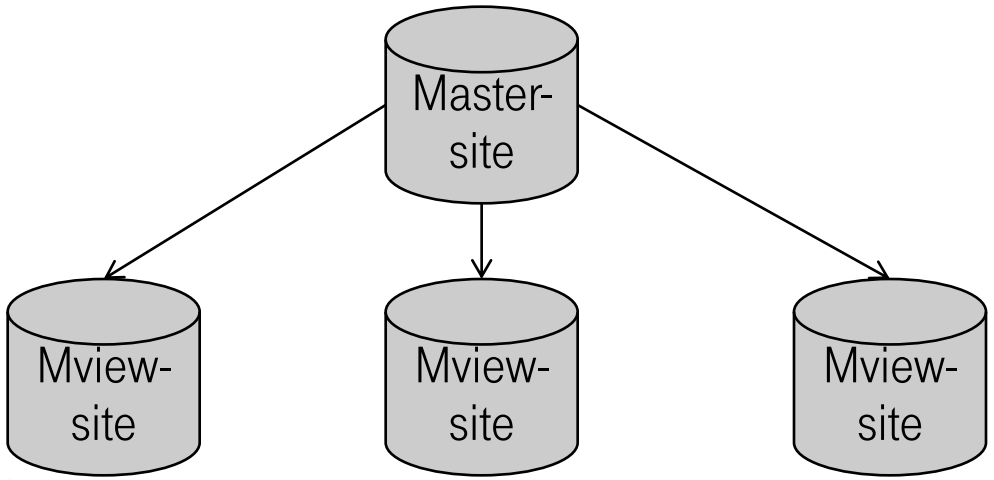
Allgemeines zur Replikation

Warum Replikation?

- Datenversorgung von externen Systemen
- Skalierung
- Viele lesende Zugriffe
- Massendatenverarbeitung auf Daten in OLTP System
- Unterschiedliche Partitionierungen

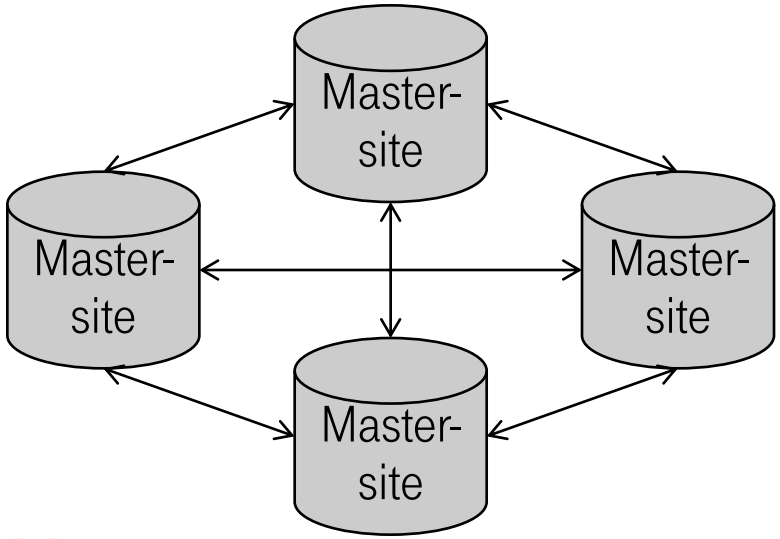
Replikationsszenarien (1/2)

- Ein System schreibt Daten, mehrere Systeme greifen lesend darauf zu (active/passive)
  - Synchron (Replikation in schreibender Transaktion)
    - Readonly Materialized View (Snapshot) mit refresh on commit
  - Asynchron (Replikation nach schreibender Transaktion)
    - Readonly Materialized View (Snapshot) mit Refresh-Job, mehrere Transaktionen werden zu einer Transaktion zusammengefasst
    - Log basiert (Oracle Streams, Golden Gate, ...), transaktionsweise Verarbeitung



Replikationsszenarien (2/2)

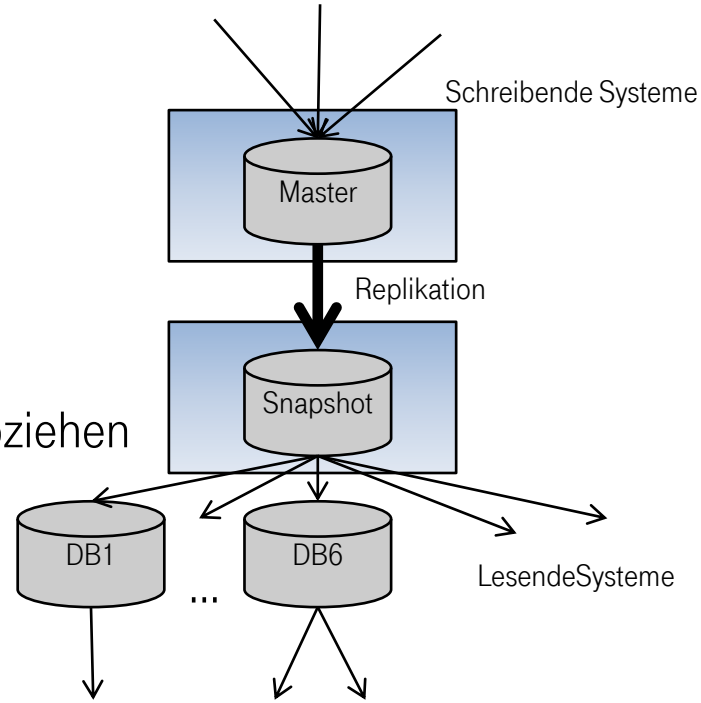
- Alle Systeme schreiben Daten (active/active)
  - Multimaster Replikation, Oracle-Streams, Golden Gate, ...
  - Konfliktauflösung ist notwendig
    - Synchron
      Konfliktauflösung in schreibender Transaktion, kann ggf. Fehler bei Doublette liefern
    - Asynchron
      Konfliktauflösung nach schreibender Transaktion, wie teilt man dass dem User mit?



T · · Systems

Betriebene 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



Materialized Views Theorie

Replikation mit Materialized Views (1/5)

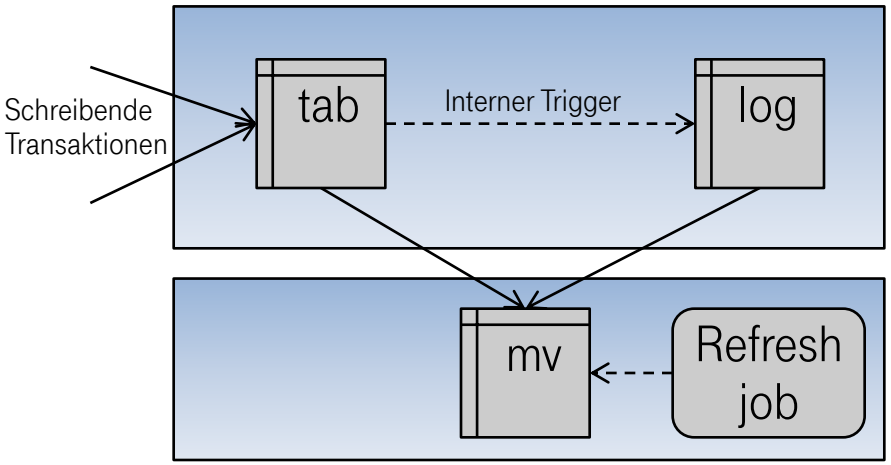
- Materialized Views sind Views mit dahinterliegender Tabelle
- Beschleunigung der Zugriffe auf die Daten
  - Keine Joins benötigt
  - Weniger Datensätze
  - Nachteil redundante Datenhaltung und somit mehr Storage

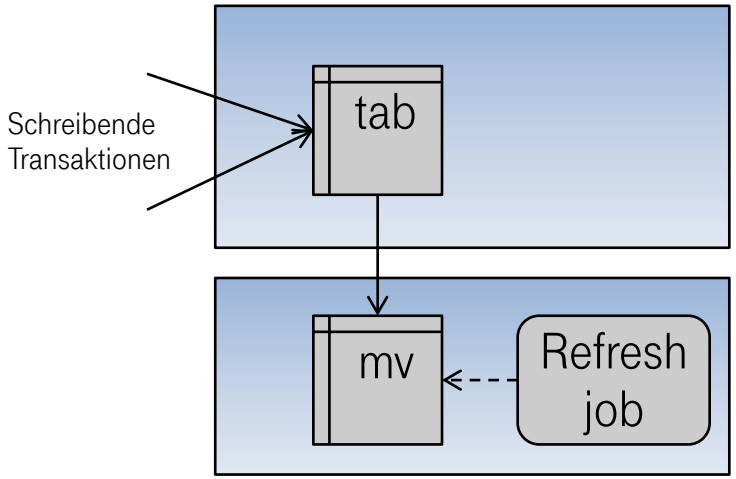
- Nutzbar
  - Lokal in gleicher DB wie Tabellen
  - Remote in anderer Datenbank wie die dahinterliegenden Tabellen

- Synchroner Refresh während COMMIT
- Asynchroner Refresh über DB-Jobs

Replikation mit Materialized Views (2/5)

- Fast Refreshable: Es werden nur die geänderten Daten transferiert. In Log-Tabelle werden neue/geänderte/gelöschte Rows protokolliert.
  - ROW_ID
  - Primary Key
- **Achtung bei Fast-Refreshable mit ROW_ID: ALTER TABLE MOVE führt bei nicht leerer Log-Tabelle zu Datenverlust im MVIEW**
- Complete Refresh: Die Tabelle wird immer komplett tranferiert





Replikation mit Materialized Views (3/5)

- Nicht jede MVIEW ist fast Refreshable
  - Nicht deterministische Funktionen (z.B. SYSDATE, RAND, ...)
  - Zu komplexe Joins
- MVIEWs können auch über eine PREBUILT TABLE angelegt werden.
  - 2 Schritte:
    - CREATE TABLE
    - CREATE MATERIALIZED VIEW
  - Beachten: Datentypen der Spalten müssen passen

- Die Datenkonsistenz von mehrere Materialized Views über Refreshgruppen
  - Refreshgruppe beschreibt Konsistenzgruppe
  - Tabellen in verschiedenen Gruppen sind ggf. nicht immer konsistent zu einander.
  - Refresh der Tabellen einer Gruppe erfolgt nacheinander in einer Transaktion.

Replikation mit Materialized Views (4/5)

Examples:

- Materialized View Log:
  - `CREATE MATERIALIZED VIEW LOG ON xxx$ta_xxx WITH PRIMARY KEY;`
- Materialized View:
  - `CREATE DATABASE LINK dblink CONNECT TO aaaa IDENTIFIED BY „bbbb“ USING `dbname`;`
  - ```
    CREATE MATERIALIZED VIEW xxx$mv_xxx  REFRESH FAST
    AS SELECT * FROM xxx$ta_xxx@DBLINK
    ```
- Materialized View on Prebuild Table
  - `CREATE TABLE xxx$mv_yyy (id NUMBER NOT NULL, …);`
  - ```
    CREATE MATERIALIZED VIEW xxx$mv_yyy ON PREBUILT TABLE REFRESH FAST
    AS SELECT * FROM xxx$ta_yyy@DBLINK
    ```

T · · Systems

Replikation mit Materialized Views (5/5)

Examples:

- Erzeuge Refreshgruppe mit 1h Refresh-Intervall
  - `DBMS_REFRESH.MAKE(`xxx$rv_g1`,`xxx$mv_xxx,xxx$mv_yyy`,SYSDATE, `SYSDATE+1/24');`
- Erweitere Refreshgruppe um eine MVIEW
  - `DBMS_REFRESH.ADD(`xxx$rv_g1`,`xxx$mv_zzz`);`
- Lösche MVIEW aus Refreshgruppe
  - `DBMS_REFRESH.SUBTRACT(`xxx$rv_g1`,`xxx$mv_zzz`);`
- Manueller Refresh:
  - `DBMS_REFRESH.REFRESH(`xxx$rv_g1`);`
  - `DBMS_MVIEW.REFRESH(`xxx$mv_xxx`,`F`);`
- Prüfen Mview der Funktionalitäten ($ORACLE_HOME/rdbms/admin/utlxmv.sql installieren)
  - `DBMS_MVIEW.EXPLAIN_MVIEW(`xxx$mv_xxx`);`
  - `SELECT * FROM MV_CAPABILITIES_TABLE;`

T · · Systems

Soweit die Theorie

Eigentlich ist Replikation mit Materialized Views recht simpel.

- Zur Nutzung muss man nur folgende Schritte abarbeiten
  - Konzeptionell
    - Welche MVIEWs müssen untereinander konsistent sein (Refreshgruppen)
    - Wie alt dürfen die Daten einer Gruppe werden (Refreshzyklen)
  - DDLs auf Mastersite:
    - MVIEW-Logs anlegen
  - DDLs auf MVIEW-Site
    - Database link anlegen
    - MVIEWs anlegen
    - Refresh-Jobs anlegen (je Refreshgruppe einer, Interval ist Refreshzyklus)

T · · Systems

Materialized Views Praxis
Installation

Installation von großen MVIEWS (1/2)

- Laufzeit wird bestimmt durch
  - Größe der MVIEWS
  - Netzwerkbandbreite (bei 1GBit Netzwerk gehen maximal ca. 80MB/Sec).
  - Latency muss bei Auswahl des Übertragungsprotokolles beachtet werden

- Parallel Refresh hat zwar Parameter PARALLELISM
  `EXECUTE DBMS_MVIEW.REFRESH(LIST=>'tab1',PARALLELISM=>8, METHOD=>'C');`
  der wirkt aber nicht wie erwartet
- MVIEW kann mit Parallel-Hint angelegt werden, dadurch wird aber nur die Query auf Master-Site parallelisiert
- Steigerung der Bandbreite durch zusätzliche Netzwerkinterfaces funktioniert nicht so wie erwartet (nur eine SQL*Net-Verbindung).

Installation von großen MVIEWS (2/2)

- Complete Refresh auf Tabelle, die parallel produktiv genutzt wird, scheitert meist am UNDO
- exp/imp (expdp/impdp) verursacht grosse temporäre Dateien. Dies kann mit Pipes und Remote-Shell optimiert werden, ist aber fehleranfällig.
- Partitionierte Tabellen können partitionsweise parallel kopiert werden (ggf. auch über verschiedene Netzwerkinterfaces – mehrere DB-Links), dies setzt MVIEWS auf PREBUILT TABLES voraus.
- Bei Initialfüllung keine Indexe auf MVIEW-Site (oder Indexe auf UNUSABLE setzen), create(rebuild) der Index nach dem Complete Refresh notwendig

- Achtung: Ein COMPLETE REFRESH startet per Default mit
  `DELETE FROM mview.`
  Das dauert beim 2. Versuch sehr lange (erzeugt viel UNDO/REDO). Mit
  `DBMS_MVIEW.REFRESH(…, method=>'C', atomic_refresh=>false)`
  kann der DELETE durch TRUNCATE ersetzt werden.

T · · Systems

Installation in Produktion

- Beobachtung: Beim Kopieren der Daten vom Master- zur MVIEW-Site sinkt der Durchsatz
  - Ursache sind Datenblöcke, die mit offener Transaktion auf Disk geschrieben wurden (z.B. bei Massenupdates). Hier muss beim Lesen ggf. zuerst ein UNDO gemacht werden (Prüfen, ob Datensätze schon commited wurden)
  - Deferred Block Cleanouts bereinigen diese Datenblöcke beim ersten Zugriff
  - Bei Zugriffen über DB-Links wird kein Deferred Block Cleanout gemacht

- Lösung: Vor dem Abziehen der Daten einen Deferred Block Cleanout mit einem Full-Table-Scan erzwingen.
  Falle: Parallel Query führt auch keinen Deferred Block Cleanout aus
- Zusätzliche Falle mit 11g: Statt SCATTERED READs macht ein Full-Table-Scan jetzt DIRECT READs.
  Direct Reads machen aber auch keinen Deferred Block Cleanout.
  Lösung: alter session set events '10949 trace name context forever, level 1';

T · · Systems

Materialized Views Praxis
Monitoring

Monitoring von Materialized Views (1/2)

- Im produktiven Betrieb von MVIEWs stellen sich immer die Fragen
  - Sind die MVIEWs aktuell, waren die MVIEWs gestern Mittag aktuell?
  - Wann wurden die MVIEWs auf eine Tabelle das letzte Mal aktualisiert?

- MVIEW-Site
  - Sind MVIEWs aktuell?

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

  - Für welche Refreshgruppen läuft gerade der Refresh?

```
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);
```

- Oracle hat kein Monitoring der Anzahl der replizierten Datensätze und der Refresh-Historie

Monitoring von Materialized Views (2/2)

- Master-Site
  - 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, dba_base_table_mviews b
where r.mview_id = b.mview_id;
```

  - Wieviele Daten stehen noch im MLOG von Tabelle xxx$ta_xxx?

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

T · · Systems

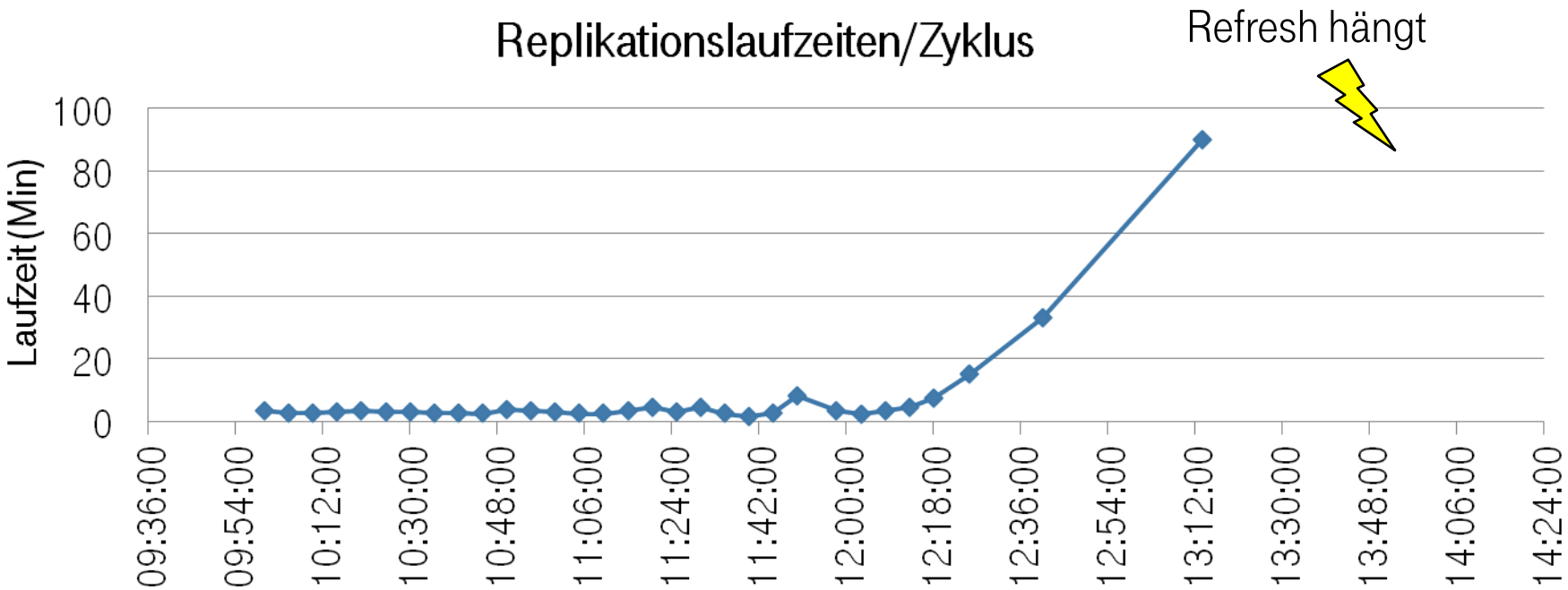
Materialized Views Praxis
Testsysteme

Bereitstellen von Testsystemen

- Beim Erzeugen von DBs als Kopien über rman/exp/imp die Zielsysteme von MVIEW-Quellsystemen so trennen, dass DB-Links von MVIEW nach Master-DB nicht funktionieren (am Besten per Firewall)
  - DB ohne JOB_Queue-Processes und ohne Scheduler starten, und erstmal alle DB-Links löschen.
  - DB-Links auf die Kopien anlegen
  - MVIEWs anpassen (hier entfällt bei PREBUILT-Tables der Datentransfer)

Materialized Views Praxis
Performanz

Performanz von Materialized View Replikation



- Im Betrieb zeigt sich obiges Bild
- Refresh-Job ist seit Stunden aktiv
- Was ist passiert?
- Spätestens jetzt muss man sich mal mit den MVIEW-Details beschäftigen.

T · · Systems

Ablauf der MVIEW-Replikation

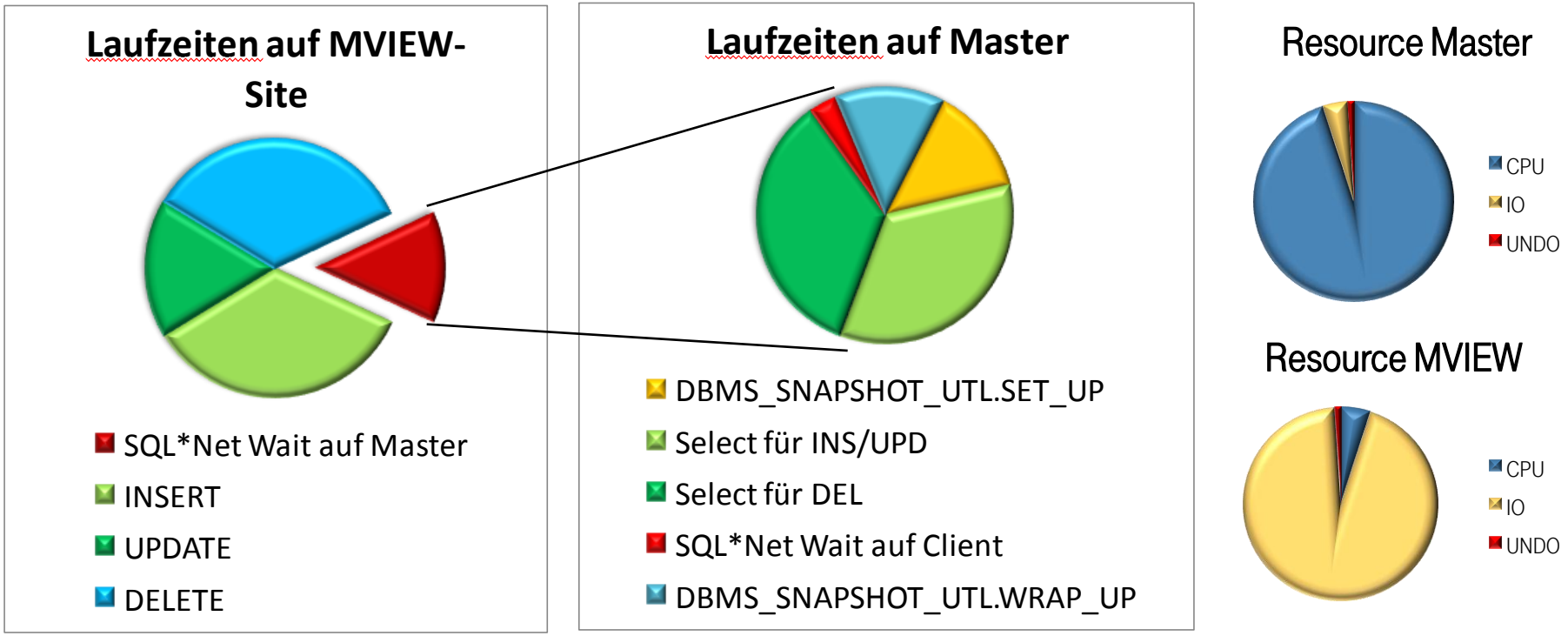
	Mastersite	MVIEW-Site
Je Replikationsgruppe		DBMS_REFRESH.REFRESH()

	`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 Replikationsgruppe		

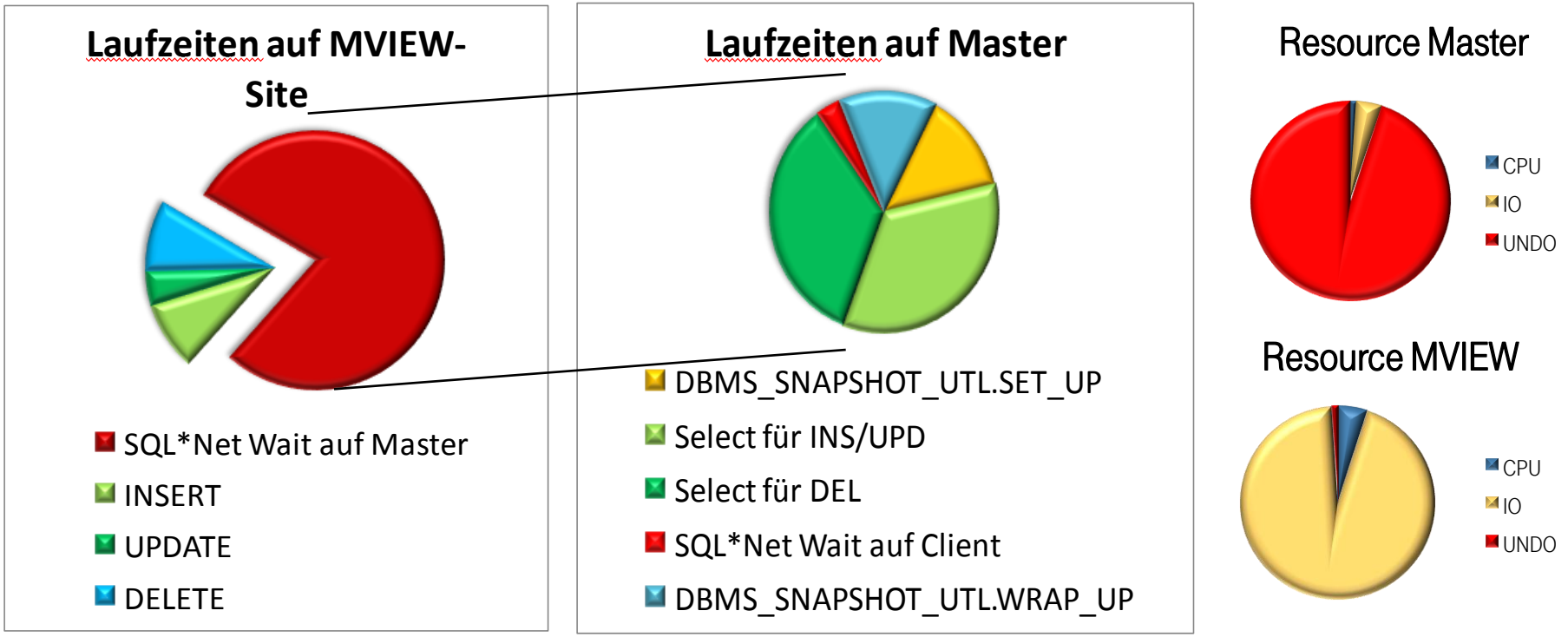
	`DBMS_SNAPSHOT_UTL.WRAP_UP`
	`UPDATE MLOG$_...`
	`DELETE MLOG$_...`
	`COMMIT;`	

Laufzeitverteilung bei „gut“ laufender Replikation



- Durchsatz wird bestimmt durch IO auf MVIEW-Site
- MVIEW-Replikation skaliert nicht

Laufzeitverteilung bei „schlecht“ laufender Replikation



- Durchsatz wird bestimmt durch UNDO auf Master-Site
- Wenn ca. 90% UNDO überschritten sind, wird Replikationsdurchsatz geringer als Änderungsrate der schreibenden Prozesse, es hilft nur noch Abbruch der Replikation.

Ursachen der „langsamen“ Replikation

- 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).
- Replikation verursacht viel IO auf Master-Site (Daten sind nicht mehr im Buffercache)
  - 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)

Performanztuning der Replikation

- 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).
- 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“ (z.B. unkritische Batchprozesse verlangsamen, damit Replikationsvolumen/Zeit immer größer als Änderungsvolumen/Zeit ist)
- Wenn eine MVIEW ausbleibt (Auszeit der DB), 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;

T · · Systems

Wenn etwas richtig schief geht

- MLOGs wachsen (Alle Refreshs der MVIEWs einer Tabelle werden langsamer)
  - Ursache: Ein MVIEW wird nicht mehr refreshed
  - Lösung:
    - Refresh des MVIEWs wieder aktivieren
    - Wenn MVIEW-Datenbank nicht mehr existiert, Unregister der zugehörenden MVIEWs
- Es fehlen Daten im MVIEW
  - Neuaufbau keine Option, da lange Auszeit
  - Delta feststellen (SELECT ... FROM xx MINUS SELECT ... from XX@DBLINK) dauert auch zu lange
  - Es wird auf Master und MVIEW ein Zeitstempel benötigt um die Menge der Rows für die Deltabestimmung einzuschränken
  - UPDATE der fehlenden Rows auf Mastersite

T · · Systems

Materialized Views Praxis
Security

Security

- Keine Public Database-Links mit CONNECT TO IDENTIFIED BY
  - Keine Zugriffssteuerung, wer was aus Remote DB sehen darf.
- Für jeden Database-Link einen eigenen Account mit minimalen Rechten nutzen.
  - Zugreifendes System ist ggf. unsicherer als man denkt.
  - Kennworte können je System getrennt geändert werden (je nach Password-Policy).
- Notwendige Zugriffsrechte für MVIEW-Zugriff auf Master-Site
  - SELECT an den zu replizierenden Tabellen
  - Bei FAST-Refreshable MVIEWS zusätzlich SELECT an den MLOG$-Tabellen.

Replikationsrelevante Oracle Bugs

Auswahl an Bugs bzgl. Materialized Views

- 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

Änderungen an Materialized Views über die Oracle-Releases

- Vor Oracle 8 hießen Materialized Views noch Snapshots
- Mit Oracle 8i wurden MVIEWs mit Spalten wie SYSDATE zu komplexen MVIEWS und somit nicht mehr fast refreshable (Es sei denn sie wurden schon vorher angelegt). Dafür gibt es seither die PREBUILT TABLE.
- Mit Oracle 9i führte ein Redesign der internen Schnittstellen dazu, dass der Datentransfer über die DB-Links massiv anstieg (Faktor >5), wurde dann mit 9.2.0.5 gefixed.
- Bis Oracle 10g machte ein DBMS_MVIEW.REFRESH (`xxx`,`C`) intern
  TRUNCATE TABLE xxx; INSERT /*+ append */ INTO XXX SELECT ....
  seit 10g per Default ein
  DELETE FROM TABLE xxx; INSERT /*+ append */ INTO XXX SELECT ....
  Das Verhalten kann mit
  DBMS_REFRESH.REFRESH(`xxx`,'C',atomic_refresh=>FALSE);
  zurückgeschaltet werden.

Tips

Tips zu Materialized View Replikation (1/2)

- Bei großen Tabellen FAST REFRESHABLE MVIEWs nutzen.
- Mit DBMS_MVIEW.EXPLAIN_MVIEW prüfen, ob wirklich FAST-Refreshable
- Nur die Zeilen und Spalten replizieren, die wirklich notwendig sind
- Replikationsgruppen so klein wie möglich.
- MVIEW-Site so sparsam wie möglich indizieren.
- Protokollierung der Refreshzyklen (Startzeit, Laufzeit, ...).
- Vermeiden großer, lang offener Transaktionen auf Mastersitze.
- PREBUILT TABLE nutzen.
- Nach Massenupdates Skrinken der MLOG-Tabellen
  `ALTER MATERIALIZED VIEW LOG ON xxx SHRINK SPACE COMPACT`
- MVIEWs mit Spalte REPLICATED_AT, Master-Tables mit Spalte MODIFIED_AT.

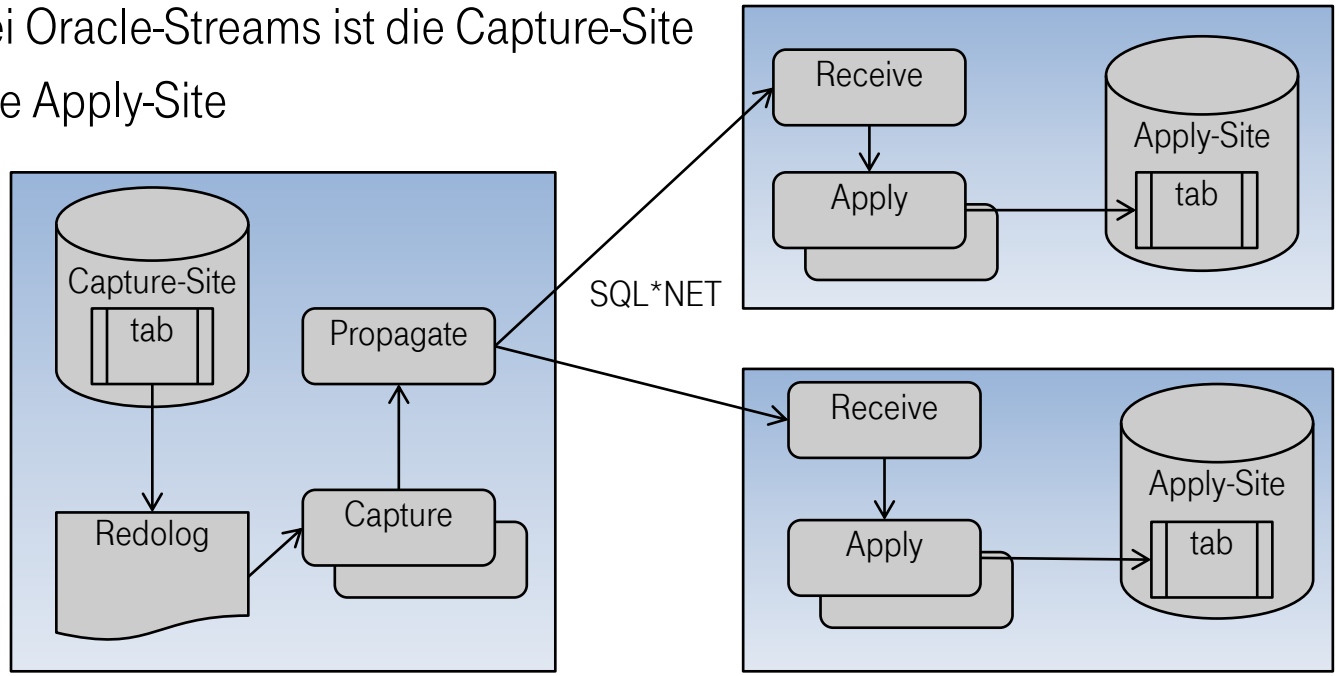
Tips zu Materialized View Replikation (2/2)

- Beim Abschalten einer DB mit MVIEWs auf Remotetabellen
  - Alle MVIEWs in dieser DB löschen, da MVIEWs sonst weiterhin in Master registriert bleiben
  - Wenn DB schon weg, DBMS_MVIEW_UNREGISTER_MVIEW zum Deregistrieren der MVIEWS nutzen

Oracle Streams Theorie

Replikation mit Oracle-Streams

- Oracle-Streams kann für unidirektionale und bidirektionale 1:n und n:n Replikation genutzt werden
- Geänderte/neue/gelöschte Rows aus den Redolog-Informationen extrahiert (LOGMINER)
- Temporäre Informationen werden im Streams-Pool gespeichert
- Quell-Datenbank bei Oracle-Streams ist die Capture-Site
- Zieldatenbank ist die Apply-Site



Vorteile Oracle-Streams

- Oracle-Streams kann beim Apply skalieren, es können beliebig viele Apply-Prozesse konfiguriert werden, jeder Prozess führt dabei eine Transaktion aus.
- Skalierung nur möglich, wenn auch entsprechend viele parallele Transaktionen laufen

- Capture kann auch skalieren, führt aber dazu, dass nur die innerhalb eines Capture-Prozesses behandelten Tabellen untereinander konsistent sind (analog MVIEW-Replikationsgruppen)

Oracle Streams Praxis

Monitoring (1/2)

- Ohne Datenänderungen sieht man Streams nicht an, ob es noch funktioniert
- Einzige Möglichkeit auf Capture-Site regelmäßig Datenänderungen (Zeitstempel) in einer Heartbeat-Tabelle, diese kann dann auf der Apply-Site auf Aktualität geprüft werden

Capture Site

- Status des Capture-Prozesses

```
select capture_name, state, total_messages_captured, total_messages_enqueued,
capture_message_number, enqueue_time, enqueue_message_create_time,
enqueue_message_number, capture_message_number-enqueue_message_number
enqueue_backlog, current_scn-capture_message_number capture_backlog,
state_changed_time, capture# capture, sid, serial#
from v$streams_capture, v$database
```

- Status

```
select capture_name, status, status_change_time, error_number, error_message
from dba_capture
```

Monitoring (2/2)

Apply-Site

- Apply-Errors

```
select apply_name, error_creation_time, error_number, error_message,
message_number, message_count, local_transaction_id
from dba_apply_error
order by error_creation_time desc
```

- Status der Apply-Server

  select apply_name, server_id, state, total_messages_applied,
  applied_message_create_time, apply_time, apply# apply, sid, serial#
  from V$STREAMS_APPLY_SERVER order by 1,2

- Status Apply-Coordinator

```
select apply_name, state,  hwm_message_create_time create_time,
hwm_time hwm_time, round((hwm_time -hwm_message_create_time) *86400)
start_delay,  round((sysdate -hwm_message_create_time) *86400) curr_delay,
total_applied, total_wait_deps, total_wait_commits,  total_rollbacks,
total_errors,
apply# apply, sid, serial#
from v$streams_apply_coordinator
```

T · · Systems

Erfahrungen (1/3)

- Plan war die Readonly-Snapshots durch Oracle Streams zu ersetzen
- Als1. Schritt wurde dafür ein Schema produktiv auf Oracle-Streams umgestellt.
- Installation ist deutlich aufwendiger (und fehleranfälliger) als mit MVIEWs

- Man kann sehen, dass die Apply-Prozesse skalieren
- Das Suplemental-Logging erhöht das Redolog-Volumen merkbar (Transaktionsabhängig)

- Eine DB mit Oracle-Streams-Replikation eines Testschemas lief über 1Jahr ohne Probleme (bis zum Upgrade der Zieldatenbank), die Datenbank hatte relativ kleines Redo-Volumen, so konnte der Capture-Prozess immer wieder aufholen.

Erfahrungen (2/3)

Probleme in Produktion:

- Wenn Capture Prozess hängenbleibt, hat er bei hoher Redo-Last (1GB/Min) kaum eine Change wieder aufzuholen. Wenn Capture/Propagate über Disk läuft, sinkt der Durchsatz sehr deutlich.
- Capture-Prozess nutzt Shared-Pool/Streams-Pool, gibt es hier Probleme, führen diese schnell zu Hängern.
- Bei Oracle-Streams-Problemen ist deren Lösung recht komplex
- Apply-Errors machen viel Arbeit, für jeden Fehler
  - Ermittlen was nicht geklappt hat, korrigieren
  - DBMS_APPLY_ADM.EXECUTE_ERROR ausführen
- Eine NOLOGGING-Transaktion und Streams muss ggf. manuell synchronisiert werden, unbedingt FORCE_LOGGING auf DB-Ebene aktivieren.
- Resync ist auch mit Oracle-Streams bei großen Tabellen im laufenden Betrieb nicht mehr möglich (DBMS_COMPARISION hilft da auch nicht wirklich)

T · · Systems

Erfahrungen (3/3)

- Datenmigrationen dauern durch FORCE_LOGGING ggf. deutlich länger
- Hängendes Capture führt dazu das RMAN Archivelogfiles nicht mehr löscht.

Testsysteme

- Da Streams ebenfalls Database-Links benutzt, gilt bzgl. Database-Links das Gleiche wie bei MVIEW-Replikation
  - Nach DB-Kopie zuerst Oracle-Jobs bzw. Scheduler deaktivieren
  - Database Links droppen
  - Streams-Konfiguration deinstallieren
  - Neue DB-Links anlegen
  - Jobs, Scheduler aktivieren
  - Streams-Konfiguration neu installieren

T-Systems

Tips zu Oracle-Streams-Replikation

- Unbedingt in der Datenbank FORCE_LOGGING aktivieren und NIE !!!! abschalten.
- Soll Oracle-Streams in mehreren Umgebungen (Entwicklung, Test/ Integration, Abnahme/Produktion) installiert werden, muss die Installation in Skriptform vorliegen. Grid-Control-hilft da nur bedingt, am Besten aus Datenmodell generieren.
- Für die Behandlung auftretender Fehler muss ein entsprechendes Skripting erstellt werden.
- Bei Systemen mit kontinuierlich sehr hohem Redolog-Volumen fehlen ggf. die Zeiten in denen Streams einen Backlog abbauen kann, Daten sind dann auf Apply-Site über längere Zeit nicht aktuell.

- Bei „normalem“ Redolog-Volumen scheint Oracle-Streams stabil zu laufen.

Weiterführende Links

Materialized Views

- Concepts, Architecture, Beispiele
  http://docs.oracle.com/cd/E11882_01/server.112/e10706/repmview.htm
- Advanced MVIEWS
  http://docs.oracle.com/cd/E14072_01/server.112/e10810/advmv.htm

Oracle-Streams

- White Paper
  http://www.oracle.com/technetwork/database/features/data-integration/twp-streams-11gr1-134658.pdf
- Streams-Doku-Einstieg
  http://docs.oracle.com/cd/E11882_01/server.112/e17069/toc.htm
- Beispiele mit DDLs
  http://docs.oracle.com/cd/E14072_01/server.112/e12862/toc.htm

T · · Systems

Vielen Dank.

Fragen und Antworten