Downtime-Minimierung für SAP-Schema-Migrationen mit Oracle Features

Andreas Prusch, Benjamin Gampert, Hendrik Müller
pasolfora GmbH
Thalmässing

Schlüsselworte

Migration, SAP, Downtime-Minimierung, Tuning, Undo, Parallelisierung

Einleitung

SAP-Schemata und andere große Datenbank-Schemata werden auch heute noch oft logisch übertragen. Dabei fließt ein sehr großer Aufwand in die Performance Optimierung dieses Vorgangs um die Ausfallzeit möglichst gering zu halten. Wir zeigen in unserem Vortrag die Möglichkeiten zur Performance Optimierung dieses Vorgehens und stellen dabei die von Oracle zur Verfügung gestellten Mechanismen und Bordmittel in den Fokus. Insbesondere geht der Vortrag auf die Bereiche Parallelisierung (Auto-DOP), LOB-Tuning und Tuning von generellen Datenbankeinstellungen ein und zeigt die möglichen Performance-Gewinne dazu auf. Angerissen werden dabei Themen wie SAP-Schema-Migrationen unter der Prämisse "Big Endian" nach Linux, wechseln des Charactersets sowie die zugehörige Characterlength-Semantik, LOB Transfer Tuning, optimale Einstellungen der Oracle Datenbanken für Migrationen (mit Fokus auf Redo und Undo) und generelles Tuning der Migration. Am Ende werden kurz Tools angesprochen, die diese Techniken heute Nutzen, wie z. B. Data Pump, O2O und HiPAS.

Optimierungen bei der logischen Datenübertragung

Einleitend werden wichtige Datenbankparameter genannt, welche für das Tuning im Vorfeld einer Migration auf Datenbankebene von Bedeutung sind.

Tuning des I/O-Verhaltens

Datenmigrationen, die auf logischer Datenübertragung basieren, erfordern eine hohe Anzahl von Schreiboperationen. Oracle pflegt Change-Vektoren für Redo- und Undo-Operationen im Logstream. Die folgenden Einstellungen sorgen dafür, dass die Server (Shadow)-Prozesse, welche die Migration ausführen, nicht auf die I/O Rückmeldung des Log Writers warten müssen (nowait) und dass der Log Writer Commits in einem Schreibvorgang zusammenfassen kann:

```
commit_logging = batch
commit_wait = nowait
db_writer_processes=20
```

Damit im Buffercache jederzeit genug freie Puffer zur Verfügung stehen, kann die Anzahl der DB-Writer wie im vorangehenden Beispiel erhöht werden.

Tuning des Caches

Für Sortierungen und Verarbeitungspuffer sind die Werte `pga_aggregate_target` und `_pga_max_size` relevant. Daten-Sortierungen profitieren außerdem von `_pga_max_size`, da Oracle auch bei `pga_aggregate_target` Grenzen für die einzelnen Prozesse, weit unterhalb des

in `pga_aggregate_target` angegebenen Wertes setzt. Die Memory-Parameter `sga_target`, `db_cache_size` und `shared_pool_size` sind vor allem auf der Zielmaschine wichtig, um die Ergebnisse aufnehmen zu können und vor allem bei Indexaufbauten so wenig wie möglich nachlesen zu müssen. Unter Linux ist der Wert `use_large_pages='only'` bei großen Caches nahezu zwingend.

Segment Tuning

Bei Tablespaces, deren allocation_type auf „system“ gesetzt ist, tritt insbesondere zu Beginn extreme High-Watermark (HWM) Contention auf. Grund ist das zunächst geringe Wachstum der physikalischen Tablespace-Allokierung. Hier sollte das Segment vor-allokiert werden.

Weitere Tuningmaßnahmen

```
event='44951 TRACE NAME CONTEXT FOREVER, LEVEL 1024'
```

Dieser Werte erhöht die High-Watermark in 1MB Schritten.

```
recyclebin='off'
```

Recycle Bin kann bei mehrfachen Versuchen sehr hinderlich sein, weil gedroppte Segmente hier Platz belegen.

```
resource_manager_plan=''
```

Wird am Wochenende migriert, sollte man die zu diesem Zeitpunkt standardmäßig laufenden Resource Manager Pläne abstellen, da diese die CPU belasten. Außerdem sollten die Oracle Autotasks in den Maintenance Windows für die Migration gestoppt werden.

```
commit_point_strength = 10
```

Da einige Daten über Datenbank-Links übertragen werden, sollte die Produktion bei diesen verteilten Transaktionen der Koordinator sein. Damit schützt man diese vor Problemen im Bereich dba_2pc_pending.

Undo-Management

Seit der Einführung mit der Datenbankversion 9i hat sich das automatische UNDO-Management etabliert und den administrativen Aufwand der DBAs verringert. Während in früheren Versionen die Rollbacksegmente von Hand erstellt, gesized und zum benötigten Zeitpunkt den mitunter langlaufenden Transaktionen zur Verfügung gestellt werden mussten, kann diese Aufgabe nun automatisiert von der Datenbank übernommen werden. Seit Version 11g ist dies auch der Default. Mit der Einstellung `undo_management=AUTO` erstellt die Datenbank selbständig Undo-Segmente im dafür konfigurierten Undo-Tablespace. Diese werden hauptsächlich für den Rollback von Datenänderungen, dem Sicherstellen von Lesekonsistenz und auch dem Transaktions- bzw. Datenbank-Crashrecovery verwendet. Mit dem Parameter `undo_retention` kann zudem definiert werden, wie lange inaktive (varaltete) Undo-Informationen für Lesekonsistenz vorgehalten werden sollen, bevor diese wieder von neuen Transaktionen überschrieben werden dürfen. Hier handelt es sich aber nur einen Richtwert, welcher sich u. a. von folgenden Faktoren beeinflussen lässt.

Retention Guarantee

Mit dieser Einstellung auf Tablespace-Ebene lässt sich der bei `undo_retention` spezifizierte Wert garantieren und führt dazu, dass keinerlei Undo-Informationen unterhalb dieser Zeitgrenze überschrieben werden. Hiermit wird die Lesekonsistenz für Queries mit höherer Priorität behandelt als etwaige aktive Transaktionen, die Platz im Undo-Tablespace benötigen.

Autoextensible vs. Fixed Size Tablespace

Viele DBAs behalten gerne die Kontrolle über das Tablespace-Wachstum und schalten die automatische Erweiterung der Datafiles aus. Beim Undo-Tablespace hat dies direkten Einfluss auf das Verhalten des automatischen Undo-Managements. Während bei Undo-Tablespaces mit fester Größe die gewählte `undo_retention` ignoriert und immer die, basierend auf Systemlast und prozentualer Tablespace-Größe, beste Einstellung getroffen wird, können beim Undo-Tablespace mit Erlaubnis zum autoextend weitere Undo-Segmente hinzugefügt werden, bevor alte Undo-Segmente überschrieben werden.

Probleme des Auto-Tunings

Obwohl das automatische Tuning der Undo-Retention in den meisten Fällen sehr gut funktioniert, kann es in Datenbanken mit hohem und wechselndem Workload zu Contention im Bereich der Undo-Segmente kommen. Dies äußert sich in den Wait-Events „latch: row cache objects“, hier speziell auf dem Child-Latch „dc_rollback_segments“, und „enq: US – contention.“

Wie bereits erwähnt, handelt es sich bei der Angabe der undo_retention um einen Minimalrichtwert, der von der Datenbank aus den bereits genannten Faktoren unterschritten, aber auch überschritten werden kann, um die bestmögliche Einstellung für die aktuelle Workload zu erzielen. So können sehr lang laufende Queries die Autotuning-Einstellung der Retention in die Höhe treiben (siehe Spalte `tuned_undoretention` in den Views `v$undostat` und `dba_hist_undostat`) und den Platz im Undo-Space aufbrauchen. Erfordert die Workload nun wieder (sehr) viele Undo-Segmente für aktive Transaktionen, können die Undo-Segmente mit hoher Retention für lange Zeit nicht überschrieben werden. Es müssen weitere Undo-Segmente online geschaltet werden, was zur Contention führen kann, wenn dies innerhalb kürzester Zeit geschehen muss. Ist es nicht möglich, weitere Segmente online zu nehmen, z. B. aufgrund von Undo-Tablespaces mit fester Größe und vollständiger Belegung des Undo-Spaces, so muss die Retention für die inaktiven Undo-Segemente wieder herabgesetzt werden. Diese Umorganisation kann mit der Workload dann meist nicht Schritt halten. Um die Contention zu verringern kann mit dem folgenden Underscore-Parameter eine maximale Undo-Retention definiert werden, nach der die inaktiven Undo-Segmente wieder zum Überschreiben freigegeben werden:

```
_highthreshold_undoretention
```

Dieser Parameter ist verfügbar ab Version 10.2.0.5 und kann online gesetzt werden. Die folgende Tabelle führt weitere Parameter auf, die das Undo-Verhalten beeinflussen können:

Tabelle 1: Parameter für das Undo-Management

`_rollback_segment_count`	gibt die Anzahl der Undo-Segmente an, die online gehalten werden sollen
`_undo_autotune" = false`	schaltet das automatische Undo-Tuning ab
`_smu_debug_mode`	(Version < 10.2.0.5) Ändert die Berechnungsart der tuned_undoretention bei fixed size Undo Tablespaces

Um Abbrüche oder Undo-Engpässe während Migrationen zu vermeiden, empfiehlt es sich eigens für die Migration große Undo-Tablespaces mit `autoextend` anzulegen. Der Undo-Tablespace kann nach der Migration auf dem Ziel wieder durch einen kleineren Tablespace ersetzt werden. Alle logischen Migrationsmethoden (Data Pump, O2O, HiPAS, etc.) sind abhängig von Undo und würden im Falle einer auftretenden Contention unter einer längeren Downtime leiden oder teilweise abbrechen. Migrationen stellen sich auf der Quelle als hohe Leselast bzw. auf dem Ziel mit vielen parallelen Transaktionen dar.

Konvertierung

Unicode

Die Zieldatenbank kann in UTF8 angelegt werden. Transfer Jobs, die Daten zwischen unterschiedlich kodierten Tabellen übertragen, reagieren auf die unterschiedlichen Zeichensätze und konvertieren korrekt. Da sich aber die Feldlängen vergrößern, sollte auf „Character Length Semantic“ umgestellt werden. Hierzu kann der Datenbankparameter `nls_length_semantics='BYTE'` eingesetzt werden. Dies wird auch von SAP so empfohlen.

Datentyp “Long”

Spalten des Datentyps `long` lassen sich nicht über DB-Links übertragen. Diese sollten im Vorfeld in LOB-Typen umgewandelt werden (`dbms_redefinition`) oder können mit der Funktion `to_lob` umgewandelt werden.

Parallelisierung

In Memory Parallel Query

Insbesondere beim Indexaufbau kann die automatische Parallelisierung (In Memory Parallel Query) eingesetzt werden. Dazu muss im Vorfeld die folgende Prozedur aufgerufen werden:

```
dbms_resource_manager.calibrate_io
```

Danach sollten über den View `dba_rsrc_io_calibrate` die Werte überprüft werden. Hier scheint das Verhältnis von `MAX_MBPS` und `MAX_PMBPS` wichtig. Wird für den Single-Prozess-Wert (PMBPS) nahezu der gleiche Werte erreicht wie für alle Prozesse gleichzeitig (MBPS), dann wird nur sehr wenig parallelisiert. Hier sollten die Werte dann manuell auf ungefähr auf 1:10 gesetzt werden.

Ein weiterer Vorteil für die Parallelisierung ist das Vorhandensein von guten Statistiken. Diese können von der Quelle wie folgt übertragen werden:

```
DBMS_STATS.EXPORT_SCHEMA_STATS
DBMS_STATS.IMPORT_SCHEMA_STATS
```

Muss die Statistik wegen Versionswechsel aktualisiert werden, kann dazu die Prozedur `DBMS_STATS.UPGRADE_STAT_TABLE` verwendet werden.

Der Wert `parallel_degree_policy=auto` ist zwingend für die Session zu verwenden, andernfalls kann „In-Memory Parallel Query“ nicht verwendet werden. Danach wird mit `parallel_min_time_threshold` der Zeitwert bestimmt, ab dem eine Parallelisierung stattfindet. Der Optimizer erstellt anhand der Statistiken einen Ausführungsplan und schätzt daraufhin ab, ob dieser Plan länger als `parallel_min_time_threshold` Sekunden dauert. Dauert der erstellte Plan länger, wird parallelisiert. Sind bereits mehr Server zur parallelen Ausführung belegt, als

in parallel_servers_target definiert, wird das Statement in eine First-In-First-Out (FiFo) Queue gestellt, womit ein Overloading verhindert wird. Ist die in parallel_servers_target definierte Anzahl von Servern noch nicht erreicht, wird das Statement gestartet. Erreicht die Parallelisierung eine höhere Prozessanzahl, als mit dem Wert parallel_max_servers vorgegeben wird die Parallelisierung verringert.

Das einzelne Statement wird außerdem nie stärker parallelisiert, als in parallel_degree_limit definiert. Darüber hinaus hat sich in unseren Tests herausgestellt, dass parallel_force_local im RAC bei herkömmlicher Architektur ein insgesamt besseres Verhalten zeigt. Da zudem bei SAP Migrationen per se genug Objekte zur Verfügung stehen, welche gleichzeitig bearbeitet werden können, sollten die RAC Instanzen für die Parallelisierung mit verschiedenen Objekten verwendet werden. Möglicherweise schafft hier Exadata mit dem „Infiniband Interconnect“ Abhilfe.

Manuelle Parallelisierung

Oracle bietet eine elegante Möglichkeit zur manuellen Parallelisierung über die dba_extents. Hier kann über „RowID“ Ranges parallelisiert werden, indem insert-select Befehle auf Teilbereiche der zu kopierenden Tabelle begrenzt werden:

```
insert /*+ NOAPPEND */ into xx select * from xx@db_link  where rowid
between 'a' and 'b';
```

Wenn auf diese Art parallel über verschiedene Sessions kopiert wird, ist zwingend mit NOAPPEND (Default) zu verfahren, da es sonst zu HWM Contentions kommt.

Parallelisierung in einer Session

Innerhalb einer Session kann eine Parallelisierung über folgende Befehle realisiert werden:

```
alter session enable parallel dml;
alter session force parallel dml parallel 40;
```

In Vorbereitung auf den Datentransfer kann anschließend eine leere Tabelle nach dem Vorbild einer existierenden Tabelle erzeugt werden:

```
create table apr3.x2 as select * from apr3.APR$LD$CLOB3@APR$MIGLINK
where 1=0;
```

Für die neue Tabelle können nologging und ein Parallelisierungsgrad vorgegeben werden:

```
alter table apr3.x2 nologging parallel 20;
```

Der folgende Insert kopiert die Daten einer Quelltabelle über einen Datenbank-Link in die neu erstellte Tabelle mit dem Parallelisierungsgrad 20:

```
insert /*+ append parallel(t1 20) */ into apr3.x2 t1 select /*+
full(t3) parallel(t3 20) */ *  from apr3.APR$LD$CLOB3@APR$MIGLINK
t2;
```

Die beschriebene Art der Parallelisierung funktioniert jedoch für LOBs nur bedingt gut, weshalb diese, wie in Abschnitt „Konvertierung“ beschrieben, zuvor umgewandelt werden müssen.

Migrationstools

Für Oracle Datenbankmigrationen steht eine Vielzahl unterstützender Tools zur Verfügung. Die hier fokussierten Offline-Migrationen lassen sich weiterhin wie in Tabelle 2 dargestellt klassifizieren:

Tabelle 2: Überblick ausgewählter Migrationsverfahren

Migrationsverfahren	Ebene/ Granularität	Downtime Proportionalität	Platform-wechsel	Endianness-wechsel	Zeichensatz-wechsel
Storage Replication	Storage/ Storage	vernachlässigbar	nein	nein	nein
Transportable Database	OS/ Datenbank	Datenbankgröße	ja	nein	nein
Transportable Tablespaces	OS/ Tablespace	Tablespace Gtöße	ja	nein	nein
Cross Platform Transportable Tablespace	OS/ Tablespace	Tablespace Größe	ja	ja	nein
Transportable Tablespaces using Cross Platform Incremental Backups	OS/ Tablespace	Datenänderungsrate	ja	nein	nein
Oracle-to-Oracle (O2O)	OS/ Schema	Menge der Migrationsdaten	ja	ja	nein
Data Pump	Datenbank/ Zelle	Menge der Migrationsdaten	ja	ja	ja
Export/Import	Datenbank/ Zelle	Menge der Migrationsdaten	ja	ja	ja
HiPAS	Datenbank/ Zelle	Menge der Migrationsdaten	ja	ja	ja

Es folgt eine kurze Vorstellung der Verfahren Data Pump (Expdp/Impdp), O2O und HiPAS.

Expdp/Impdp

Starke Verbesserungen der herkömmlichen exp/imp Utilities führten zum aktuellen Werkzeug Data Pump (expdp/impdp). Data Pump arbeitet im Gegensatz zum exp/imp mittels Hintergrundprozessen (Data-Pump-Worker-Prozesse), deren Anzahl durch den Parameter `parallel` eingestellt werden kann. Die Worker-Prozesse werden über einen Master-Prozess je Data Pump Job koordiniert. Die Views `dba_datapump_jobs` und `user_datapump_jobs` zeigen eine Zusammenfassung zu allen aktiven Data Pump Jobs innerhalb des Systems an bzw. solche, die vom aktuell angemeldeten Benutzer ausgelöst wurden. Data Pump lässt starke Parallelisierung zu. Bei großen LOBs kann die Performance erfahrungsgemäß jedoch einbrechen. Im Normalfall wird ein Zwischenspeicher benötigt. Data Pump lässt sich aber auch zu Lasten der Performance im Netzwerkmodus über Datenbank-Links nutzen.

O2O

O2O (Oracle2Oracle) ist ein universelles Tool für Oracle Migrationen im SAP Umfeld, mit dem kurze Migrationszeiten erreicht werden können. Es erfordert jedoch einen vergleichsweise hohen manuellen Tuning-Aufwand. O2O besteht aus einem PL/SQL-Package, das auf dem Quellsystem ausgeführt wird und aus einem Scheduler, der auf dem Zielsystem aus der Betriebssystemebene heraus operiert. Dieser beherrscht unterschiedliche Übertragungsmechanismen (darunter auch Data Pump und die Verwendung von Datenbank-Links) und kann je nach Vorgabe parallelisieren. Long-Datentypen von bis zu 32k können in VARCHAR2-Spalten überführt werden. Loginformationen müssen innerhalb des Dateisystems auf OS-Ebene identifiziert und analysiert werden.

HiPAS

HiPAS (High Performance Adaptive Schema Migration) ist eine Datenbank-Migrationssoftware, die Oracle-Features nutzt, um die Downtime zu minimieren. Durch adaptive Parallelisierung ist die Software in der Lage, sich kontinuierlich den verfügbaren Hardware-Ressourcen anzupassen, um das System jederzeit optimal zu belasten. Ein Optimizer kontrolliert hierzu eine Vielzahl an Monitoring-Informationen wie Concurrency Events, durchschnittliche Schreib- und Lesezeiten, die Größe des Redo Log Buffers und weitere. Daraufhin wird, wenn erforderlich, der Parallelisierungsgrad angepasst. Um eine flexible Adaption zu ermöglichen, wird die Gesamtmenge der zu migrierenden Daten in gleich große Transfer Jobs aufgeteilt und über alle zur Verfügung stehenden Instanzen und Datenbank-Links verteilt. Die Software arbeitet vollständig innerhalb der Datenbankebene, die Migration erfolgt „on the fly“ ohne dass ein Anlegen von Export- oder Logdateien erforderlich ist. Log-Informationen und andere Metadaten werden stattdessen in temporären Tabellen gespeichert. Long-Datentypen erkennt und konvertiert HiPAS automatisch auf dem Quellsystem, bevor diese übertragen werden.

Kontaktadresse:

Andreas Prusch

pasolfora GmbH

An der Leiten 37

D-91177 Thalmässing

Telefon: +49 (0) 170-3440 650

Fax: +49 (0) 9173-7932 91

E-Mail andreas.prusch@pasolfora.com

Internet: www.pasolfora.com