ORACLE IN-MEMORY: EINSATZANALYSE AUS EINEM KUNDENPROJEKT

Mila Friedman
Lufthansa Systems GmbH & CO.KG
Raunheim

Schlüsselworte

Oracle in-Memory (IMDB), Linux VM, Komprimierung, SSD Storage, PCI Express, Optimierung.

Einleitung:

Die bestehende Real_Time DWH läuft auf der Oracle 11.2.0.4 Datenbank und hat die Größe von 2TB. Die Datenbank besteht aus einer Fact Table mit den aktuellen Daten, den Basis Tabellen und vielen Aggregates mit den entsprechenden historischen Daten. Für eine bestehende Applikation wurde nach einer neuen Lösung gesucht, die nicht nur performant und zukunftssicher, sondern welche auch die weiteren „Flexible Reportings“ ermöglichen kann. Hierbei wurde beschlossen die Oracle In-Memory Technologie zu untersuchen.
Die Oracle In-Memory Option sollte in definierten repräsentativen Tests eine Performanceverbesserung gegenüber der aktuellen Umgebung bei gleicher Konfiguration ohne Applikationsänderung aufzeigen.

In diesem Vortrag werden Erfahrungen und kritische Aspekte im Hinblick auf den Einsatz der Oracle In-Memory Technology vorgestellt und alternative Lösungen diskutiert.

Oracle In-Memory POC

Die für diesen POC ausgewählten Reportingabfragen stellen eine kleine Auswahl dessen dar, was durch den Kunden zur Verfügung gestellt wird.

Für ein Oracle in Memory POC wurde eine neue '12.1.0.2.0' Datenbank mit einem Datenset in der Größe von 200GB auf folgendem VM Server aufgebaut:

LINUX VM

Red Hat Enterprise Linux Server release	6.5	
CPU	16	Cores
RAM:	450	GB
Hugepagesize	2048	KB

Bei der Aktivierung von Hugepages muss berücksichtigt werden, dass die Parameter MEMORY_TARGET und MEMORY_MAX_TARGET ausgeschaltet werden sollten.

Wie auch andere kostenpflichtige Optionen der Enterprise Edition ist die In-Memory-Option standardmäßig aktiviert. Zusätzlich dazu muss für „in-Memory Column Store“ ein Speicher allokiert werden. Dafür wird der Parameter `inmemory_size` (per Default `inmemory_size=0`) entsprechend gesetzt. Da „In-Memory Column Store“ ein Bestandteil von SGA ist, muss SGA_TRAGET groß genug konfiguriert werden. Im Anschluss muss die Datenbank neu gestartet werden, um den In-Memory Bereich in SGA zu initialisieren. Wenn es notwendig ist, kann man die Memory Option mit `inmemory_query=disable` wieder deaktivieren.

Im nächsten Schritt muss man eine Memory Option für alle betroffenen Objekte (Tabelle/Spalte/Partition/Tablespace/Materialized View) mit dem Kommando z.B. `alter table <table_name> inmemory` eischalten und die Objekte werden in die neue In-Memory Area geladen.
Ein großer Vorteil hierbei gegenüber einigen anderen In-Memory Datenbkanken Z.B. HANA ist, dass nicht der komplette Datenstack im Column-Store Format gespeichert werden muss. Darüber hinaus können die Objekte mit unterschiedlichen Ladeprioritäten (CRITICAL, HIGH, MEDIUM, NONE(default) konfiguriert werden. Hierbei ist es wichtig zu berücksichtigen, dass mit NONE Lade-Priorität konfigurierte Objekte nur beim ersten Zugriff in den in-Memory Bereich geladen werden. Ebenfalls können die Komprimerunsverfahren unterschiedlich gewählt werden (NO MEMCOMPRESS, MEMCOMPRESS for DML/FOR QUERY LOW(default)/ FOR QUERY HIGH/ FOR CAPACITY LOW/ FOR CAPACITY HIGH). Aber auch hier sollte man berücksichtigen, dass bei MEMCOMPRESS FOR CAPACITY HIGH/LOW die Daten bei der Bearbeitung zunächst entkomprimiert werden. Unter Berücksichtigung der genannten Aspekte würden ist es deshalb empfehlenswert, nur die selten benutzten Objekte mit NONE Lade-Priorität und MEMCOMPRESS FOR CAPACITY LOW/ HIGH zu definieren.

Beim Laden in die In-Memory Area zerlegt das System die Tabellen in Spalten und komprimiert sie mit einem unterschiedlichen Faktor. Damit werden die beiden Konzepte: die klassische Speicherung: `ROW BASED` und die neue `COLUMN BASED` in einer Datenbank integriert. Die Ladezeit hängt hierbei von der Datenmenge, der Datenverteilung und dem `INMEMORY_MAX_POPULATE_SERVERS` (default 0,5 * CPU_COUNT)
Parameter ab. Auch die große Anzahl von Chained Rows wirkt sich negativ auf die Ladezeit aus.

Nicht mehr benötige Tabellen/Partitionen können jeder Zeit aus dem IM-Memory Store mit dem Kommando herausgenommen werden: alter table <table_name> no inmemory. Es ist hierbei wichtig, in v$im_segments immer zu prüfen, ob alle betroffenen Tabellen/Partitionen in IM-Column store erfolgreich geladen wurden.

```
 1  select v.segment_name name,partition_name, v.INMEMORY_PRIORITY,v.populate_status status, INMEMORY_SIZE/1024/1024/1024 INMEMORY_SIZE_GB ,
 2* v.bytes_not_populated/1024/1024 NOT_POPULATED_MB,v.bytes/v.inmemory_size comp_ratio   from v$im_segments v order by  v.bytes_not_populated/1024/1024 desc

NAME                           PARTITION_NAME                 INMEMORY_PRIORITY        STATUS                        INMEMORY_SIZE_GB NOT_POPULATED_MB COMP_RATIO
------------------------------ ------------------------------ ------------------------ ----------------------------- ---------------- ---------------- ----------
                               PART_123                       CRITICAL                 COMPLETED                           3.72766113       16857.4844 14.0472214
                               PART_124                       CRITICAL                 COMPLETED                           3.78527832        7287.1875 11.3564449
                               SEG_INFOS_201409               CRITICAL                 COMPLETED                            .07208252       938.609375 14.7671465
                               SEG_INFOS_201403               CRITICAL                 COMPLETED                           .033630371       231.546875 8.18874773
                               SEG_INFOS_201408               CRITICAL                 COMPLETED                           .740966797             64.5 1.43657331
                               SEG_INFOS_201406               CRITICAL                 COMPLETED                           .716491699        63.515625 1.48564614
                               SEG_INFOS_201405               CRITICAL                 COMPLETED                           .740966797        39.609375 1.35222405
                               SEG_INFOS_201407               CRITICAL                 COMPLETED                           .745056152         8.828125 1.32383059
                               PART_46                        CRITICAL                 COMPLETED                           .126342773                0 5.87439614
                                                              CRITICAL                 COMPLETED                           2.76104736                0 2.48999713
                               PART_46                        CRITICAL                 COMPLETED                           .218139648                0 3.97537773
                               ABHIST_201311                  CRITICAL                 COMPLETED                            .22479248                0 1.74640239
                               PART_48                        CRITICAL                 COMPLETED                           .001098633                0 1.77777778
                               PART_132                       CRITICAL                 COMPLETED                           .678771973                0 10.2436831
                               HUB_SEG_201506                 CRITICAL                 COMPLETED                           .073486328                0 1.94019934
                               PART_40                        CRITICAL                 COMPLETED                           .013916016                0 35.3684211
                               HUB_SEG_201309                 CRITICAL                 COMPLETED                           .067504883                0 1.99638336
                               PART_51                        CRITICAL                 COMPLETED                           .110656738                0   6.283508
                               HUB_MKT_201403                 CRITICAL                 COMPLETED                           .092102051                0 1.63286945
                               SALES_201307                   CRITICAL                 COMPLETED                           .333984375                0 1.47953216
                               SEG_INFOS_201409               CRITICAL                 COMPLETED                           .344177246                0 1.59460897
```

Wenn nämlich nicht alle Daten in Memory geladen werden konnten, kommt in Alert log leider keine Fehlermeldung und muss man anschließend in v$inmemory_area prüfen, ob in IM Column Store noch ausreichend Platz vorhanden ist. Es sollte ebenfalls berücksichtigt werden, dass Oracle ca. 20-25% von INMEMORY_SIZE für die Verwaltung des InMemory Caches verwendet.

```
SQL> show parameter inmemory_size;

NAME                                 TYPE                              VALUE
------------------------------------ --------------------------------- ------------------------------
inmemory_size                        big integer                       50G
[ sys @ T121M ]:
SQL> select sum(bytes/1024/1024/1024) ORIGINAL_SIZE_GB,sum(INMEMORY_SIZE/1024/1024/1024) INMEMORY_SIZE_GB from v$im_segments;

ORIGINAL_SIZE_GB INMEMORY_SIZE_GB
---------------- ----------------
      288.387695       40.1728516

[ sys @ T121M ]:
SQL>  select pool,alloc_bytes/1024/1024/1024 allocated_GB,  used_bytes/1024/1024/1024 used_GB, POPULATE_STATUS from V$INMEMORY_AREA;

POOL                                       ALLOCATED_GB    USED_GB POPULATE_STATUS
------------------------------------------ ------------ ---------- ------------------------------------------------------------------
1MB POOL                                     39.9589844 39.9541016 DONE
64KB POOL                                    9.97906494     .21875 DONE
```

Zusätzlich ist uns aufgefallen, dass leere Partitionen in der v$im_segments nicht angezeigt werden.

Die ersten Tests haben zeigt, dass die Abfragen nicht schneller, sonders langsamer geworden sind. Dies lag daran, dass obwohl die Tabellen im Memory lagen, ist ein Tabellenzugriff immer noch über „TABLE ACCESS BY INDEX“ stattgefunden hat.

```
 31 |                 HASH JOIN                             |                              |   127 |  6350 |    48   (78
 32 |                  HASH JOIN                            |                              |   127 |  5334 |    47   (79
 33 |                   TABLE ACCESS INMEMORY FULL          |                              |   100 |  1500 |    14   (65
 34 |                   TABLE ACCESS INMEMORY FULL          |                              |   966 | 26082 |    33   (85
 35 |                  TABLE ACCESS INMEMORY FULL           |                              | 10508 | 84064 |     1    (0
 36 |                 TABLE ACCESS INMEMORY FULL            |                              | 10508 | 84064 |     1    (0
 37 |                TABLE ACCESS BY INDEX ROWID BATCHED    |                              |     1 |    52 |    28    (0
 38 |                 INDEX RANGE SCAN                      |                              |   167 |       |     1    (0
 39 |               TABLE ACCESS BY INDEX ROWID BATCHED     |                              |     5 |    75 |     1    (0
 40 |                INDEX RANGE SCAN                       |                              |     5 |       |     0    (0
 41 |              TABLE ACCESS BY INDEX ROWID              |                              |     1 |    14 |     2    (0
 42 |               INDEX UNIQUE SCAN                       |                              |     1 |       |     1    (0
 43 |             TABLE ACCESS BY INDEX ROWID BATCHED       |                              |     6 |   132 |    12    (0
 44 |              INDEX RANGE SCAN                         |                              |    12 |       |     2    (0
 45 |            TABLE ACCESS BY GLOBAL INDEX ROWID         |                              |     1 |    32 |     0    (0
 46 |             INDEX UNIQUE SCAN                         |                              |     1 |       |     0    (0
 47 |           TABLE ACCESS BY INDEX ROWID                 |                              |     1 |    11 |     1    (0
 48 |            INDEX UNIQUE SCAN                          |                              |     1 |       |     0    (0
 49 |          TABLE ACCESS BY GLOBAL INDEX ROWID           |                              |     1 |   379 |     0    (0
 50 |           INDEX UNIQUE SCAN                           |                              |     1 |       |     0    (0
 51 |         TABLE ACCESS BY INDEX ROWID BATCHED           |                              |     1 |    39 |     1    (0
 52 |          INDEX RANGE SCAN                             |                              |     4 |       |     0    (0
 53 |        TABLE ACCESS BY GLOBAL INDEX ROWID BATCHED     |                              |     1 |    64 |     1    (0
 54 |         INDEX RANGE SCAN                              |                              |     1 |       |     0    (0
 55 |       INDEX RANGE SCAN                                |                              |     3 |       |     2    (0
 56 |      TABLE ACCESS BY GLOBAL INDEX ROWID               |                              |     1 |   881 |     5    (0
                                                                                                                       105,13
```

Nur nachdem alle analytischen Indizes auf „invisible" gesetzt worden sind, hat Oracle-Optimizer für Zugriff über Memory „TABLE ACCESS INMEMORY FULL" sich entschieden und die Tests konnten fortgesetzt werden. Obwohl auf dem Server noch genug Memory zur Verfügung stand, wurden alle HASH Operationen sofort auf den TEMP Tablespace verlagert, Auch die Konfiguration von hash_area_size und diversen Hidden Parameters: _smm*,_pga*,optimizer_mjc_enabled, _optimizer_sortmerge_join_enabled, _optimizer_sortmerge_join_inequality konnte die Verlagerung auf TEMP nicht stoppen. Nach 2 Stunden waren im TEMP Tablespace bereits 220GB belegt, was ein großes Problem darstellte. Ergänzend wurde während der weiteren Analyse festgestellt, dass die Storagezugriffe zu diesem Punkt extrem langsam waren.

```
Top 10 Foreground Events by Total Wait Time
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

                                          Total Wait       Wait   % DB Wait
Event                               Waits Time (sec)    Avg(ms)   time Class
------------------------------ ---------- ---------- ---------- ------ --------
direct path write temp             509,494      10.5K      20.62   72.6 User I/O
DB CPU                                          3562.4              24.6
PX qref latch                    2,549,112       64.6       0.03     .4 Other
library cache: mutex X                 104       11.1     106.41     .1 Concurre
latch free                             313       10.3      32.95     .1 Other
latch: parallel query alloc bu         289        8.7      29.97     .1 Other
cursor: pin S                           18        2.9     159.07     .0 Concurre
control file sequential read           945        1.5       1.59     .0 System I
db file sequential read                 71         .5       6.99     .0 User I/O
latch: shared pool                      46         .4       8.50     .0 Concurre
```

Um beide Probleme zu lösen, wurde die Entscheidung getroffen, die Abfragen zu optimieren und gegebenenfalls nachgelagert in einer anderen Umgebung zu testen. Auch wenn durch Optimierung einer Abfrage die Laufzeiten auf ein Drittel verkürzt wurden, wurden die Zeiten für das Real-Time Reporting nicht akzeptabel. Der Einsatz von EXADATA war leider nicht möglich, da bei den durchgeführten POC nach einer Lösung gesucht wurde, die in der Infrastruktur von der Lufthansa Systems integriert werden konnten. Weitere Tests wurden auf dem Server mit einem SSD Storage mit PCI Express für TEMP Tablespace durchgeführt. Bei allen Abfragen wurde die Ausführungszeit drastisch verkürzt. Bei weiteren Tests wollten wir dem Kunden die Möglichkeit geben, die historischen Daten von den letzten 24 Monaten direkt zu analysieren und ein „Flexible Reporting" durchzuführen. Für diese Tests wurde eine neue FACT* Tabelle aufgebaut, die nicht nur die aktuellen sondern auch die historischen Daten von den letzten 6 Monaten enthielt. Bei den Abfragen wurden die Daten aggregiert, gruppiert und sortiert. Die Abfragen wurden mit unterschiedlichen Konfigurationen durchgeführt. Bei allen Abfragen gegen solche FACT* Tabellen wurde nur durch Einschalten der In-Memory Option eine signifikante Verbesserung erreicht.

FAZIT:

Vor der Entscheidung für den Einsatz von Oracle In-Memory, müssen die wichtigsten/zeitintensivsten Statements der Applikation untersucht werden. Ergibt diese Untersuchung, dass Oracle In-Memory eine geeignete Lösung sein kann, muss darauf geachtet werden, dass Temp Tablespace auf schnellem Storage liegt. Erscheint Oracle In-Memory keine sinnvolle Alternative für eine Applikation zu sein, kann einerseits durch die Optimierung des Statements und andererseits durch die sinnvolle Veränderung der Infrastruktur ein starker Performanceschub erzielt werden.

Kontaktadresse:
Mila Friedman
Lufthansa Systems GmbH & CO.KG
Am Prime Park 1
D-654791 Raunheim
Telefon: +49 (0) 69-696 6277
E-Mail mila.friedman@lhsystems.com