In-Memory-Computing mit der Oracle-Datenbank 11g R2

Matthias Weiss, ORACLE Deutschland B.V. & Co. KG

In-Memory-Verarbeitung ist in aller Munde. Unternehmen erhoffen sich Verarbeitungs- und Prozess-Beschleunigung beziehungsweise sehen dies als Voraussetzung für Industrie 4.0 und/oder Internet of Things.

Dieser Artikel zeigt In-Memory-Funktionen für die Oracle-Datenbank 11*g* R2, die aus dem regulären Support läuft, aber bei vielen Kunden noch im Einsatz ist. Sie sind schnell und einfach umzusetzen, um Performance-Gewinne zu erzielen beziehungsweise neue Geschäftsfelder zu erschließen. Außerdem werden eventuell notwendige und sinnvolle Hardware-Erweiterungen und -Ergänzungen kurz erläutert. Abschließend kommen die drei großen Erweiterungen für In-Memory-Verarbeitung der Oracle-Datenbank 12*c* (ab 12.1.0.2) kurz zur Sprache, um einen Planungshorizont zu bieten. Nachfolgend die zur Verfügung stehenden In-Memory-Funktionen:

- DB-Cache
- Row-Caches
- Statement-Caches
- Pinnen von ganzen Tabellen
- Oracle-Text-Index im Memory
- Query-Result-Cache
- In-Memory Parallel Query (RAC)
- Cache Fusion (RAC)
- Cache Hierarchie
- Smart-Flash-Cache

Um die Optimierungsmöglichkeiten für In-Memory-Verarbeitung besser einordnen zu können, muss man die Hauptspeicher-Struktur der Oracle-Datenbank verstehen. Die rot markierten Bereiche in *Abbildung 1* sind für diesen Beitrag essenziell wichtig.

Grundsätzlich sollte die Hauptspeicher-Ausstattung ausreichend sein. Die Varianz ist allerdings von Kunde zu Kunde beziehungsweise von Anwendung zu Anwendung so groß, dass es keine eindeutige Empfehlung geben kann. Allerdings sind viele Datenbank-Server, insbesondere im Mittelstand, mit einem zu kleinen Hauptspeicher ausgestattet. Ein Ausbau mit 32, 64 oder 128 GB reicht in den meisten Fällen nicht aus; sinnvolle In-Memory-Verarbeitung beginnt bei mindestens 256 GB. Bestimmte Rechner-Architekturen lassen jedoch bei Zwei-Socket-Servern nur einen maximalen Ausbau von 768 GB zu, was nicht immer ausreicht, sodass hier nach Alternativen geschaut werden muss (siehe Kapitel „Smart-Cache-Funktionalität"). Die *Abbildungen 2 und 3* verdeutlichen, wie sich Hardware-Konfigurationen durch die Nutzung von In-Memory-Techniken ändern und was beim Beschaffungsprozess neuer Server für In-Memory-Computing berücksichtigt werden muss.

Bei einer SAN-I/O-Leistung von 700 MB/s und einer durchschnittlichen Core-Leistung von 200 MB/s sind acht Cores für die optimale Verarbeitungsgeschwindigkeit notwendig, unter der Annahme, dass rund die Hälfte der Core-Kapazität für Spitzenbelastung, wie bei vielen Kunden gewünscht, als Reserve vorgehalten werden.

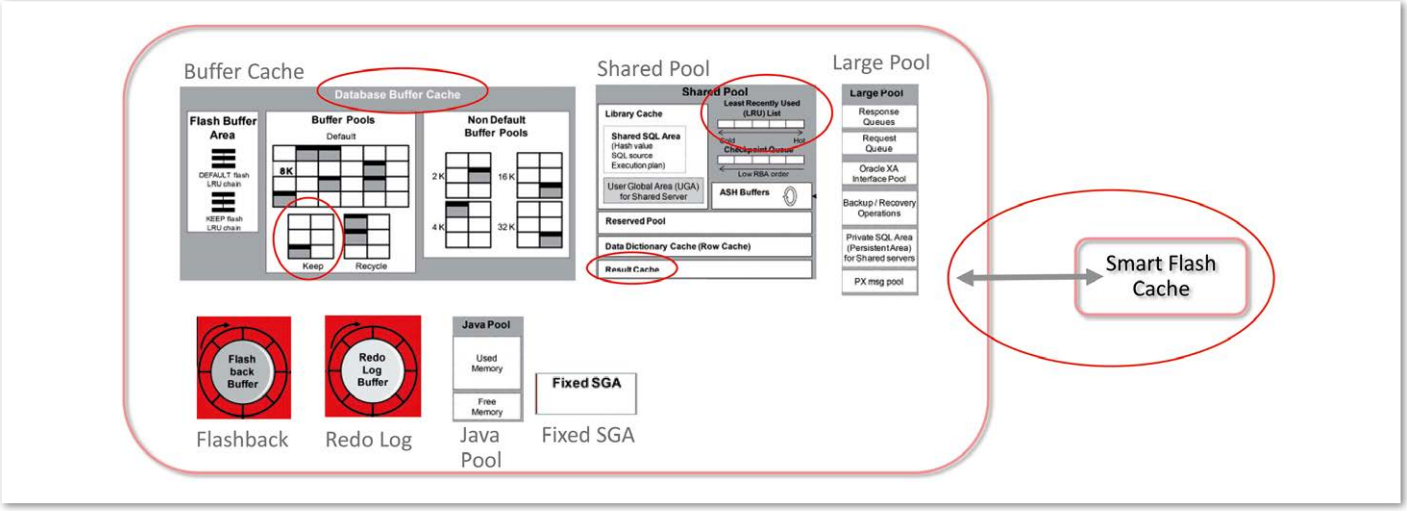

Abbildung 1: Oracle-Memory-Struktur (SGA)

Bei gleicher SAN-I/O-Leistung sind beim In-Memory-Computing deutlich mehr Cores für eine optimale Verarbeitungsgeschwindigkeit notwendig. Die Daten-Bereitstellung („Memory I/O“) liegt jetzt bei mindestens 5 GB/s. Unter der gleichen Prämisse für die Cores wie bei einer klassischen Verarbeitung (200 MB/s; 50 Prozent Auslastung) liegt die optimale Core-Anzahl jetzt bei 50.

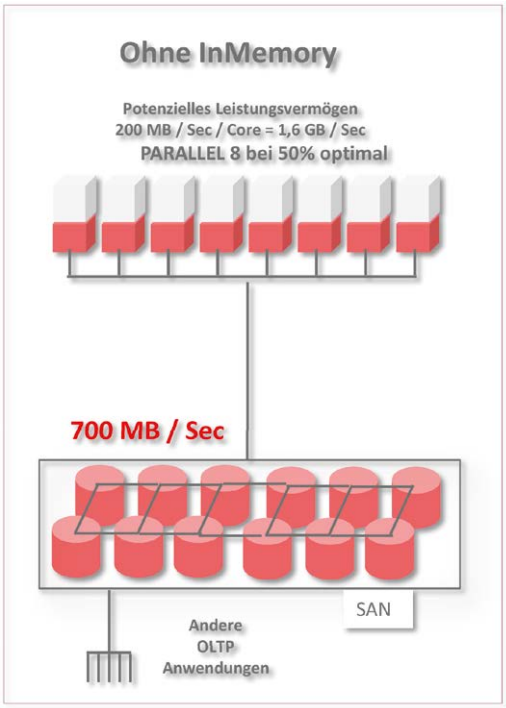

Abbildung 2: Ohne In-Memory

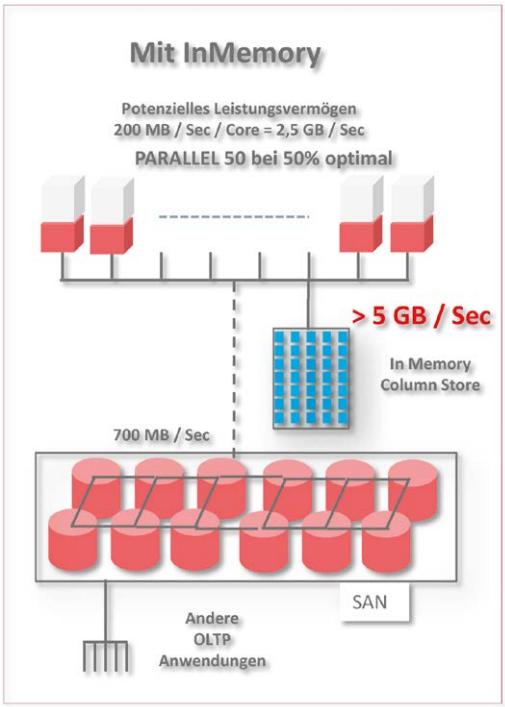

Abbildung 3: Mit In-Memory

Beim In-Memory-Computing verschiebt sich die Last vom Storage-I/O in den CPU-Bereich. Dies erfordert in vielen Fällen eine Änderung der Hardware-Konfiguration. Die Verarbeitungsgeschwindigkeit steigt dramatisch an und es lassen sich dadurch viele Geschäftsvorteile erzielen beziehungsweise bestimmte Geschäftsmodelle bis hin zur Echtzeitverarbeitung realisieren. Je größer der Datenbank-Cache, desto mehr Daten können im Hauptspeicher für die schnellstmögliche Verarbeitung in der Oracle-Datenbank liegen und desto individueller lassen sich die einzelnen Bereiche an die besonderen Anforderungen der Anwendungen und Benutzerbedürfnisse anpassen.

Daten im Buffer-Cache behalten

Die Idee beim Pinnen von Objekten ist, dass Objekte im Pool nicht verdrängt werden. Einem Pool werden nur so viele Objekte zugewiesen, bis er voll ist. Alle Segment-Typen wie LOB, INDEX etc. werden unterstützt, aber auch nicht relationale Daten wie JSON, Spatial oder XML. Dies bedeutet, dass die entsprechende Klausel „ALTER TABLE <table_name> CACHE;“ für das Pinnen von Objekten nicht den Cache benutzt, sondern die Blöcke auf den MRU-Punkt („most recently used“) der LRU-Liste („least recently used“) legt. Diese Objekte müssen aber erstmal von der Platte in den Hauptspeicher geladen werden, damit sie dann auch dort verbleiben. Dazu ist etwa Vorarbeit nötig. Der Pool ist um die entsprechende Größe des Objekts zu erweitern. Als Daumenwert gilt, dass der Pool um das eineinhalb- bis zweifache der Größe des Objekts zu vergrößern ist, also für eine Zwei-GB-Tabelle auf drei bis vier GB.

Der Buffer-Cache wird in zwei Bereiche („KEEP“ und „RECYCLE POOLS“) aufgeteilt, die vom Algorithmus her gleich funktionieren (Init-Parameter „DB_KEEP_CACHE_SIZE“ und „DB_RECYCLE_CACHE_SIZE“). Einem dieser Bereiche sind die Objekte (Tabellen, Partitionen, etc.) zugeordnet, die im Cache gehalten werden sollen. In den Keep-Pool gehören Daten, auf die häufig zugegriffen wird.

Die Bezeichnung der Pools als „Keep“, „Recycle“ und „Default“ könnte zu einem einfacheren Verständnis auch einfach „Pool1“, „Pool2“ und „Pool3“ lauten. Da Daten jedoch erst in den Cache kommen, nachdem sie erstmalig abgefragt wurden, kann bei Bedarf eine Prozedur ausgeführt werden, um eine Query zu erzwingen. Oracle legt keine Tabellen-Buffer in den Pool, bis eine Query auf diese Tabelle ausgeführt wird.

Die Vorgehensweise: Zuerst einmal ist die Pool-Größe zu definieren, hier beispielhaft der Keep-Pool über „DB_KEEP_CACHE_SIZE“; anschließend sind die Objekte per Storage-Klausel dem KEEP-Pool zuzuweisen *(siehe Listing 1)*.

Letztendlich ist je nach Bedarf ein „Pre-Load“ der Objekte in den KEEP-Pool (etwa Trigger bei „On startup database“) zu implementieren. Dazu wird ein Objekt-Scan über die Tabelle, den Index oder das Lob-Segment ausgelöst. Dies ist prinzipiell eine „SELECT“-Anweisung *(siehe Listing 2)*.

Es ist zu beachten, dass der Buffer-Cache optimiert werden muss. Dabei ist der Parameter „_small_table_threshhold“ auf größer oder gleich „benötigte Blöcke“ zu setzen. Weitere Informationen dazu am Ende des Artikels.

Einmal rechnen, mehrfach nutzen

In der Oracle-Datenbank gibt es einen Cache, der speziell für Ergebnisse aus SQL-Abfragen

```
ALTER TABLE ... STORAGE (buffer_pool keep)
ALTER INDEX ... STORAGE (buffer_pool keep)
ALTER TABLE ... MODIFY LOB (lobcol) (STORAGE (buffer_pool keep))
```

Listing 1

```
SELECT /*+ FULL(TAB) */ SUM(numeric_column), min(txt_column) FROM
tabelle TAB;
SELECT /*+ FULL(TAB) */ dbms_lob.getlength(lob_column) FROM tabelle
TAB;
```

Listing 2

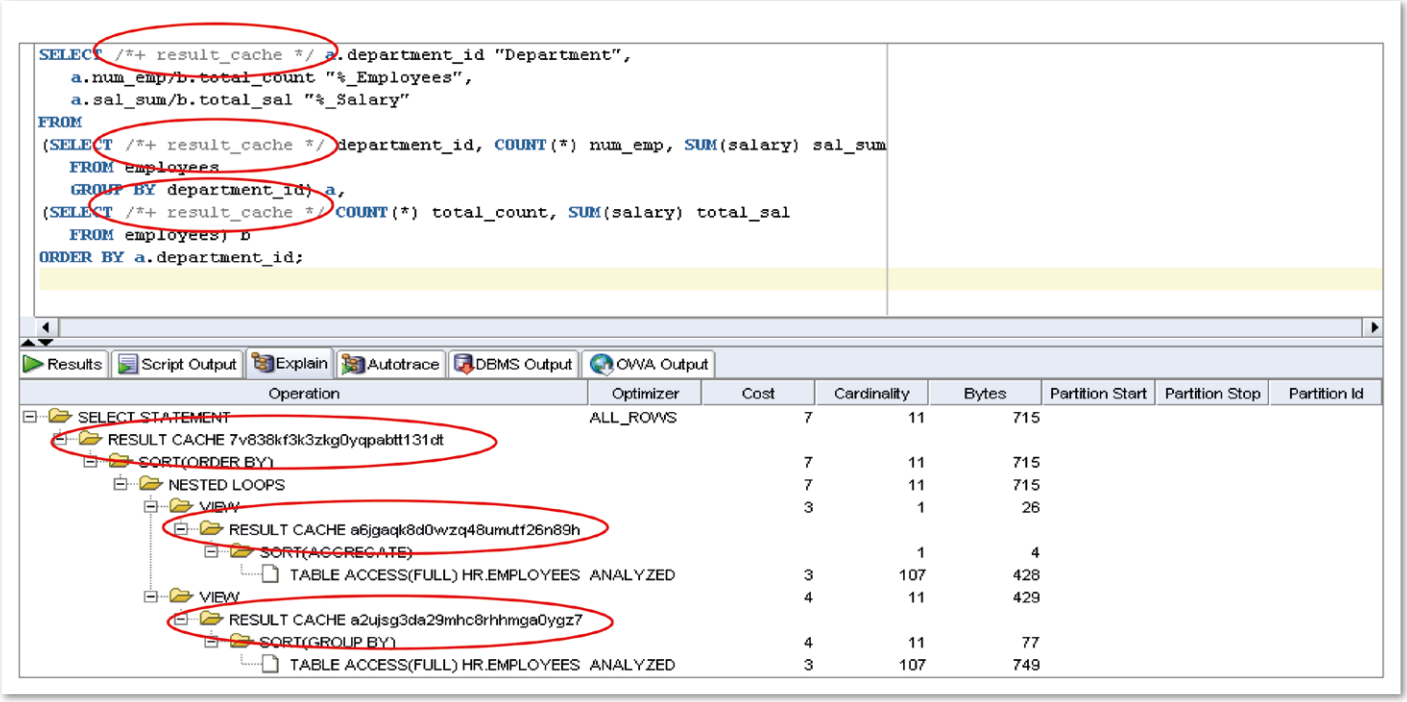

Abbildung 4: Result Cache – Query Hints

oder PL/SQL-Funktionen in der Shared Global Area (SGA) reserviert ist, der Server Result Cache. Dort sind Ergebnisse einer Datenbank-Abfrage oder ein „Query-Block" zur wiederholten Nutzung gespeichert. Der SQL Query Result Cache enthält die Ergebnisse von SQL-Queries und der PL/SQL Function Result Cache die Werte, die von einer PL/SQL Funktion stammen. Die Datenbank verwaltet den Cache völlig selbstständig und stellt dabei sicher, dass niemals veraltete Ergebnisse ausgegeben werden.

Der Oracle Call Interface (OCI) Client Result Cache ist ein Memory-Bereich innerhalb des Client-Prozesses, der SQL-Query-Ergebnisse der OCI-Applikationen speichert. Es wird empfohlen, Client-Result-Caching nur für Abfragen von „read-only"- oder „read-mostly"-Tabellen zu nutzen.

Der Entwickler muss nur entscheiden, ob man den Result Cache nutzen möchte oder nicht, und der DBA kümmert sich um einige Parameter, um alles optimal auf die Anwendungen abgestimmt bereitzustellen.

Um eine Nutzung zu ermöglichen, sind Parameter zu definieren. Dabei ist zu beachten, dass die Defaults eher zu klein und auf jeden Fall anzupassen sind. Über „ALTER SESSION", „ALTER SYSTEM" oder per Hint wird der Result Cache eingeschaltet und nutzbar. Der DBA legt durch die Parameter den individuellen Rahmen für den Result Cache fest *(siehe Tabelle 1)*. Dabei gilt:

- „RESULT_CACHE_MAX_RESULT" legt den prozentualen Anteil am Result Cache für die einzelnen Ergebnisse fest
- „RESULT_MAX_SIZE" definiert die Gesamtgröße des reservierten Bereichs im Shared-Pool in Bytes
- „RESULT_CACHE_MODE" bietet den Automatismus oder die manuelle Steuerung. Bei „MANUAL" ist explizit ein Hint erforderlich. „FORCE" ist nicht zu empfehlen, da ansonsten alle SQL-Queries und PL/SQL-Ergebnisse im Cache landen
- „RESULT_CACHE_REMOTE_EXPIRATION" legt die Anzahl der Minuten für die Validität eines Result-Set fest

RESULT_CACHE_MAX_RESULT	5 (%)
RESULT_CACHE_MAX_SIZE	abh. von O/S
RESULT_CACHE_MODE	MANUAL/FORCE
RESULT_CACHE_REMOTE_EXPIRATION	0 (min)

Tabelle 1

```
SELECT /*+ RESULT_CACHE*/ * FROM tabelle
ALTER TABLE tabelle RESULT_CACHE FORCE
ALTER SESSION SET RESULT_CACHE_MODE=FORCE
```

Listing 3

Lang laufende und rechenintensive SQL-Abfragen beziehungsweise PL/SQL-Funktionen müssen identifiziert werden. Vorhersehbare SQL-Abfragen oder gleichbleibende deterministische Ergebnismengen werden erkannt und die DML-Aktivitäten auf den zugrunde liegenden Tabellen beobachtet, damit nur Tabellen mit geringen Änderungen für die Result-Cache-Funktion genutzt werden.

Den Result Cache optimal nutzen

Die Nutzung des Result Cache *(siehe Abbildung 4)* ist durch drei Faktoren bestimmt: auf SQL-Query-Ebene durch den Hint „RESULT_CACHE", bei der Einstellung „RESULT_CACHE" in der Tabellen-Definition und durch den Session-Parameter „RESULT_CACHE_MODE" *(siehe Listing 3)*. *Listing 4* zeigt eine Beispiel-Query ohne Hin-

weise. Das Monitoring des Result Cache ist sehr einfach und der Report bietet einen einfachen und schnellen Überblick *(siehe Listing 5)*.

In-Memory Parallel Query

Die Antwortzeit eines Full-Table-Scans lässt sich durch die Nutzung mehrerer paralleler Execution-Server deutlich verbessern. Falls der Server viel Speicher besitzt, kann die Datenbank Informationen in der SGA cachen, anstatt „Direct Reads" in der PGA auszuführen. Parallel Query kann sowohl individuell und manuell eingestellt als auch automatisch genutzt werden, und das sogar über Rechner-Grenzen hinaus in einem RAC-Verbund. Zur Verfügung stehende Ressourcen werden dadurch optimal genutzt. Bei der automatischen Nutzung („Automatic Parallel Degree Policy") legt Oracle alles selbstständig fest, sowohl ob überhaupt eine Parallelisierung durchgeführt wird, als auch, welcher Parallelisierungsgrad genutzt wird *(siehe Listing 6)*.

Jedes SQL-Statement wird optimiert und durchläuft einen Parallelisierungs-Prozess beim Parsen. Wenn die Parallelisierung genutzt wird, kommen folgende Schritte zum Tragen: Die User-Session oder der Shadow-Prozess übernehmen die Rolle des Koordinators, auch „Query Koordinator" genannt. Dann beschafft dieser die notwendige und verfügbare Anzahl von parallelen Servern, damit das SQL-Statement seine Abfolge von Operationen (ein Full Table Scan, um einen Join auf „non indexed" Spalten durchzuführen, eine „ORDER BY"-Clause etc.) verarbeiten kann. Jeder Parallel-Execution-Server versucht, seine Operation parallel und – soweit möglich – unabhängig von anderen auszuführen. Wenn die Parallel-Server die Abarbeitung des Statements beenden, übernimmt der Query-Koordinator alle Arbeit, die nicht parallel erledigt werden kann, um letztendlich das Ergebnis an den Benutzer zu geben. Der SQL-Tuning-Advisor hilft bei der Analyse und Optimierung enorm. Bei diesen beiden Nutzergruppen zeigen sich deutliche Leistungsgewinne *(siehe Abbildung 5)*:

- *Allgemeiner Anwender*
  - Kleinere Datenmenge
  - Shared Pool ausreichend
  - Direct Path Read
- *Führungskraft*
  - Große Datenmenge
  - Shared Pool zu klein

Dieses Beispiel zeigt, welche Performance-Gewinne zu erzielen sind, aber dass auch andere Komponenten wie ein zu kleiner Shared Pool Auswirkungen haben. Es kommt immer auf eine abgestimmte Kombination der unterschiedlichen In-Memory-Funktionen an.

Smart Flash Cache

Wie bereits erwähnt, sind die durchschnittlichen Server mit zu kleinem Hauptspeicher für In-Memory-Computing ausgestattet. Natürlich ist eine Aufrüstung möglich, allerdings stößt man bei weitverbreiteten Servern relativ schnell an die Hardware-Begrenzungen von 768 GB pro Server bei Zwei-Socket-Maschinen. Dies ist in vielen Anwendungsfällen nicht ausreichend, insbesondere in Hinblick auf die neuen In-Me-

```
SELECT
   a.department_id           "Department",
   a.num_emp/b.total_count   "%_Employees",
   a.sal_sum/b.total_sal     "%_Salary"
FROM (
   SELECT department_id,
          COUNT(*)       num_emp,
          SUM(salary)    sal_sum
   FROM employees
   GROUP BY department_id
) a, (
   SELECT
       COUNT(*)       total_count,
       SUM(salary)    total_sal
    FROM employees
) b
ORDER BY a.department_id;
```

Listing 4

```
SQL> set serveroutput on
SQL> execute dbms_result_cache.memory_report()
R e s u l t   C a c h e   M e m o r y   R e p o r t
[Parameters]
Block Size          = 1K bytes
Maximum Cache Size  = 6M bytes (6K blocks)
Maximum Result Size  = 307K bytes (307 blocks)
[Memory]
Total Memory = 151840 bytes [0.022% of the Shared Pool]
... Fixed Memory = 5296 bytes [0.001% of the Shared Pool]
... Dynamic Memory = 146544 bytes [0.021% of the Shared Pool]
....... Overhead = 113776 bytes
....... Cache Memory = 32K bytes (32 blocks)
........... Unused Memory = 27 blocks
........... Used Memory = 5 blocks
............... Dependencies = 1 blocks (1 count)
............... Results = 4 blocks
................... SQL  = 4 blocks (3 count)
```

Listing 5

```
Manuell
ALTER TABLES sales PARALLEL 8;
ALTER TABLE customers PARALLEL 4;

Automatisch
PARALLEL_DEGREE_POLICY is set to AUTO
```

Listing 6

mory-Funktionen der Datenbank 12*c* und die Ziele von In-Memory-Computing –die Realisierung von Big Data, Echtzeit-Analysen und vielem mehr.

Der Hauptspeicher (Level 1) ist leicht durch PCI-Flashkarten (Level 2) aufzurüsten. Diese Server-Aufrüstung ist nicht mit der Flash-Erweiterung des Storage-Systems zu verwechseln, womit andere Ziele verfolgt und die besten Vorteile erst durch die Nutzung intelligenter Software erzielt werden, die auf die Datenbank abgestimmt ist, wie es bei einer Exadata der Fall ist. Folgendes ist für die Nutzung der Smart-Flash-Cache-Funktionalität nötig:

- *Hardware-Voraussetzung*
  - Flashspeicher (PCI) im Server
- *Software-Anforderungen*
  - Datenbank EE (11*g* R2 od. 12*c*)
  - Betriebssystem
    - Oracle Linux
    - Solaris

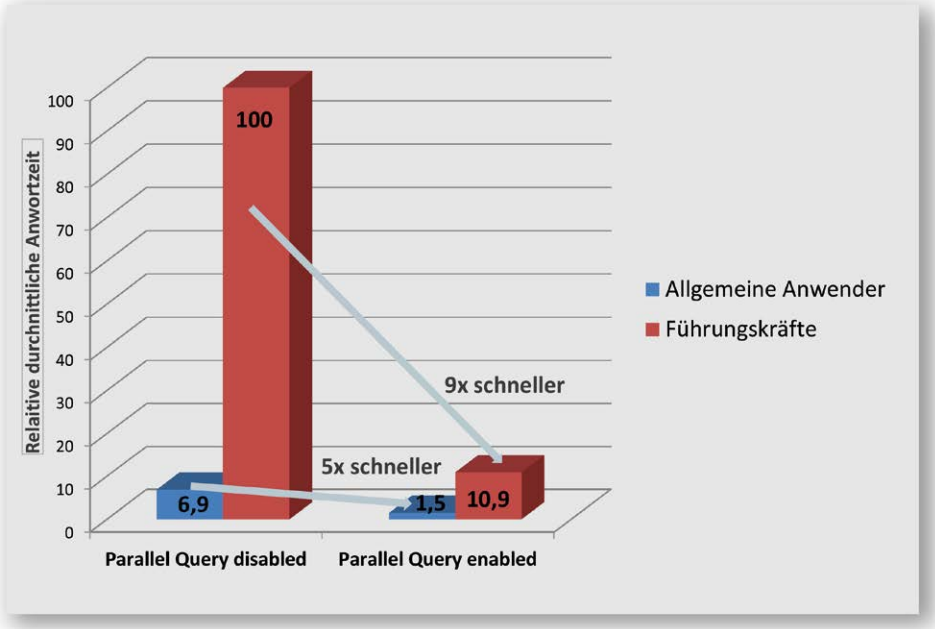

Abbildung 5: In-Memory Parallel Query

Der Smart-Flash-Cache ist eine Erweiterung des Datenbank-Buffer-Cache. Die PCI-Flashkarten sind günstiger als Memory-Komponenten, allerdings auch etwas langsamer, aber immer noch deutlich schneller als eine Platte. Außerdem bietet diese Second-Level-Cache-Erweiterung den Vorteil, dass sie eine weitaus größere Kapazität hat als der normale maximale Hauptspeicher-Ausbau der Standard-Server. Bei Lese-Operationen lässt sich ein Performance-Gewinn von bis zu Faktor „100“ gegenüber einer Disk erzielen. Dies kommt gerade den erwähnten Anwendungsfällen wie Analyse, Big Data etc. zugute. Durch diese Erweiterung und Nutzung als Buffer-Cache-Erweiterung steigt letztendlich der I/O-Durchsatz des Gesamtsystems deutlich an. Diese Konfiguration kommt in der neuen Oracle Database Appliance (ODA X5) zum Einsatz *(siehe Abbildung 6)*.

Dieses Verhalten der Datenbank bringt einen klaren Performance-Vorteil, was vielfach dokumentiert ist und durch das folgende Schaubild verdeutlicht wird. Es ist aber auch erkennbar, dass es Grenzwerte gibt, ab denen ein größerer Smart-Flash-Cache keinen Leistungsschub mehr bringt, sondern der interne Verwaltungsaufwand der Datenbank die Leistungsverbesserung übersteigt. Dazu gibt es keine allgemeingültigen Aussagen, sondern dies

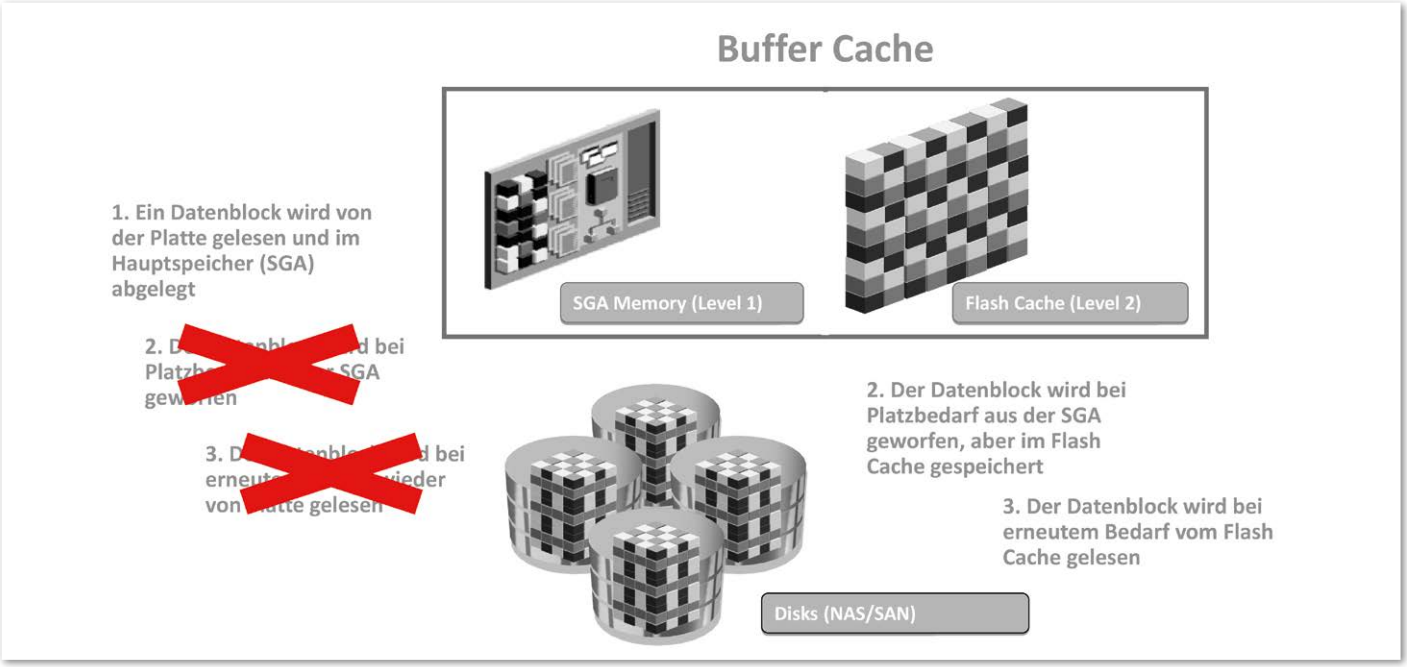

Abbildung 6: Beim Oracle-Datenbank-Verhalten mit Smart Flash Cache entfallen die Punkte 2 und 3

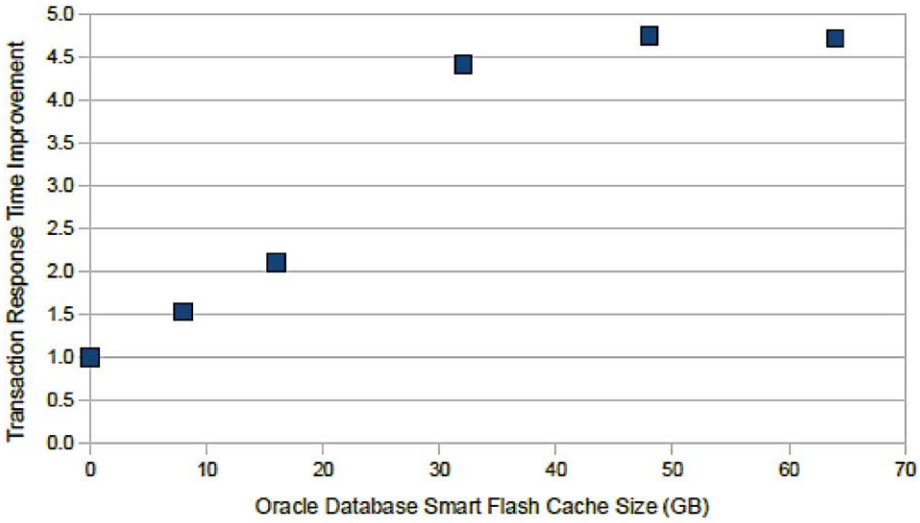

Figure 3: Response Time Improvement with Increasing Database Smart Flash Cache Size

Abbildung 7: Smart Flash Cache – Performance-Verbesserung (siehe „http://de.slideshare.net/ylouis83/11g-r2-flashcachetips“)

ist Projekt- bzw. Anwendungs-spezifisch zu klären.

Die Datenbank läuft auf Oracle Linux oder Solaris. Der Performance-Flaschenhals liegt beim Buffer-Cache; die Buffer-Pool-Advisory-Sektion des Automatic-Workload-Repository- Reports (AWR) oder der STATSPACK-Report zeigen, dass die Verdoppelung der Größe des Buffer-Cache von Vorteil wäre. Bei den Top-Wait-Events erscheint „db_file_sequential_read” und es sind freie CPU-Ressourcen vorhanden. Dazu sind die Init-Parameter „db_flash_cache_file = {OS-Pfad zur Flash Disk}“ und „db_flash_cache_size = {Größe der Flash Disk}“ einzustellen sowie die Strategien zum Pre-Loading von Objekten zu beachten. Die Storage-Klausel „FLASH_CACHE { KEEP | NONE }” wird auf Tabellen-Ebene gesetzt. Der zusätzliche SGA-Bedarf für Metadaten-Verwaltung (pro Datenbank-Block 100 Byte; auf RAC-Systemen 200 Byte) ist beim Memory-Sizing zu berücksichtigen. Wichtig zu wissen ist, dass im RAC die Flash-Disks exklusiv von einer Instanz genutzt werden.

Fazit

Die Nutzung der vorgestellten In-Memory-Funktionen für die Oracle Datenbank 11*g* R2 und das erste 12*c*-Release bringen enorme Leistungsgewinne. Der höchste Mehrwert liegt in der Kombination der Technologien und ist Datenbank-, Kunden- beziehungsweise Anwendungs-spezifisch. Ergänzt und deutlich erweitert werden diese Möglichkeiten durch neue, im Ausblick dargestellte Funktionen.

Ausblick

Mit der Oracle-Datenbank 12*c* (ab 12.1.0.2) sind die In-Memory-Funktionen der Datenbank in drei wesentlichen Punkten erweitert worden. Zwei Funktionen sind mehr oder minder unbemerkt kostenfrei im Basisumfang der Oracle-Datenbank Enterprise Edition enthalten:

- *Full DB in Memory*
  Die ganze Datenbank im Hauptspeicher zu halten, ist eine gute Möglichkeit, um I/O-Engpässe zu vermeiden. Durch die Nutzung einer Kombination von Level-1- und Level-2-Hauptspeicher für die Oracle-Datenbank (Smart Flash Cache) lassen sich eventuelle Hardware-Restriktionen einfach umgehen, etwa Zwei-Sockel-Server mit einem maximalen Memory-Ausbau von 768 GB. Oracle bietet aber auch Alternativen, bei denen Server mit bis zu 32 TB Memory ausgestattet sein können.

- *Automatic Big Table Caching*
  Datenbanken mit großen Tabellen können enorm vom Performance-Gewinn durch „Automatic Big Table Caching“ profitieren.

Besondere Beachtung findet natürlich die In-Memory-Datenbank-Option für die Version 12*c*, die eine spaltenorientierte Datenhaltung und alle von Kunden geforderten Grundfunktionen wie Sicherheit, Verfügbarkeit, Persistenz, sicheres Backup und Recovery bietet. Innerhalb eines RAC ist sogar eine Parallelisierung beziehungsweise Duplizierung des Column-Stores über Rechnergrenzen hinaus möglich. Es lassen sich Performance-Steigerungen erzielen, wie sie früher undenkbar waren. Dadurch werden die Verarbeitung und Analyse von Massendaten in nahezu Echtzeit möglich, ohne dass eine Voraggregation der Daten notwendig ist.

Ideal ist natürlich die Kombination mit den vorgestellten 11*g*-R2-In-Memory-Funktionen. Oracle hat darauf geachtet, dass die Inbetriebnahme der In-Memory-Datenbank-Option für die 12*c* ohne Migration und Neu-Implementierung nicht nur möglich, sondern so einfach wie ein Schalter zu nutzen ist – also einfach einschalten und ausprobieren.

Weitere Informationen

1. Memory Tuning: https://docs.oracle.com/database/121/TGDBA/part3.htm#sthref644
2. Smart Flash Cache: http://docs.oracle.com/cd/E11882_01/server.112/e25494/memory.htm#ADMIN13391
3. Parallel Query: http://docs.oracle.com/cd/E11882_01/server.112/e25523/parallel002.htm
4. DBACommunity Tipp: https://apex.oracle.com/pls/apex/GERMAN_COMMUNITIES.SHOW_RESOURCE_BY_FNAME?P_TIPP_ID=362&P_FILE_NAME=index.html
5. MOS Note 787373.1

Matthias Weiss
matthias.weiss@oracle.com