Toolgestützte Oracle Performance Analyse aus Consultant-Sicht

Marcus Mönnig
Lichtblick AG
Hamburg

Schlüsselworte:

Ausführungsplan, AWR, Bindevariable, Mumbai, Performanceanalyse, Performancetuning, Snapper, Statspack

Einleitung

Die Performanceanalyse und das -tuning von Datenbankapplikationen auf produktiven Kundensystemen als externer Consultant oder als Vertreter des Softwarelieferanten unterliegen einer Reihe von Einschränkungen die dazu nötigen von der reinen Lehre der Oracle-Performanceanalyse abzuweichen. Beispielsweise können Analysen oftmals nicht auf den produktiven Systemen ausgeführt werden, sind nur eingeschränkte Rechte verfügbar oder es lässt sich eine genaue Spezifizierung des Performanceproblems nicht erreichen.

Der Referent hat seine Erfahrungen in diesem Problemfeld in das Windows-Programm "Mumbai" [1] gegossen, das kostenlos im Netz zur Verfügung steht und das die effiziente Verwendung vorhandener Tools zur Performanceanalyse (AWR, Statspack, Snapper, 10046 Trace file analysis, OraSRP, Execution plan Generierung, etc.) ermöglicht.

Dieses Manuskript und der zugehörige Vortrag zeigt einige Problemfelder auf und wie Mumbai in diesen unterstützten kann.

Inhalt

1.) Wie ermittle ich den „richtigen“ Ausführungsplan?

In gebräuchlichen Tools für Datenbankadministratoren und Entwickler wird der Ausführungsplan für ein gegebenes SQL-Statement meist über das SQL Kommando "EXPLAIN PLAN" oder die SQL*Plus Autotrace Funktion (bzw. einer Nachbildung dieser) ermittelt. Beide Lösungen können dazu führen, dass bei der Performanceanalyse eines SQL-Statements mit Bindevariablen ein anderer Ausführungsplan angezeigt wird, als er unter tatsächlichen, produktiven Bedingungen genutzt werden würde oder wurde.

SQL*Plus Autotrace liefert im Allgemeinen den richtigen, tatsächlich genutzten Ausführungsplan, zeigt allerdings nicht alle für die Performanceanalyse relevanten Informationen an. Außerdem ist problematisch, dass das SQL Statement tatsächlich ausgeführt werden muss bevor der Ausführungsplan angezeigt wird.

Eine Alternative die die dargestellten Probleme löst, bietet die Verwendung des Oracle-Packages DBMS_XPLAN.

Die beiden folgenden Beispiele zeigen die Probleme der Nutzung des SQL Kommandos "EXPLAIN PLAN" bei SQL Statements die Bindevariablen enthalten.

Zunächst wird eine Testtabelle angelegt und mit 100.000 Datensätzen gefüllt:

```
CREATE TABLE MUMBAI42.t_test_data
( DATE_TWO_DISTINCT )
AS
SELECT
  case when rownum <= 10
    then sysdate
    else TO_DATE('01/01/2000 00:00:00', 'DD/MM/YYYY HH24:MI:SS')
  end
FROM
  dual
CONNECT BY LEVEL <= 100000;
```

Die Tabelle enthält nur eine Spalte vom Typ DATE und diese ist nur 10 mal mit dem aktuellen Zeitstempel und die restlichen 99.990 mal mit dem Datum des Beginns des laufenden Jahrtausends (01.01.2000) gefüllt.

Anschließend wird ein Index auf die Datumspalte erstellt und Objektstatistiken für die Tabelle und den Index erstellt:

```
CREATE INDEX MUMBAI42.t_test_data_Date2distinct
ON MUMBAI42.t_test_data (DATE_TWO_DISTINCT);

BEGIN
DBMS_STATS.GATHER_TABLE_STATS('mumbai42', 't_test_data');
END;
/
```

Im Folgenden betrachten wir nun die Ausführungspläne für das SQL-Statement:

```
SELECT * FROM MUMBAI42.t_test_data
WHERE DATE_TWO_DISTINCT = :T;
```

Prinzipiell sind zwei Möglichkeiten des Zugriffs auf die relevanten Daten denkbar:

- Zugriff über den Index auf der Spalte DATE_TWO_DISTINCT
- Zugriff über einen FULL TABLE SCAN der Tabelle

1.1) EXPLAIN PLAN und Werte von Bindevariablen

Für unser Beispiel liefert EXPLAIN PLAN auf obiges SQL Statement den folgenden Ausführungsplan mit einem FULL TABLE SCAN:

```
-------------------------------------------------------------------------------
| Id  | Operation         | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |             |   100K|  1953K|   108   (5)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| T_TEST_DATA |   100K|  1953K|   108   (5)| 00:00:01 |
-------------------------------------------------------------------------------
```

Welcher Zugriff effizienter ist, hängt (neben einer Reihe von weiteren Randbedingungen) davon ab welche Datensätze angezeigt werden sollen bzw. wie viele Datensätze in der Ergebnismenge sind. Dies ist aber abhängig vom tatsächlichen Wert der Bindevariable T.

Das EXPLAIN PLAN Kommando arbeitet ohne Bindevariablen, d.h. die Bindevariablen eines SQL Statements müssen gar nicht definiert und Wert zugewiesen werden um ein EXPLAIN PLAN Kommando absetzen zu können.

Wie viel Aussagekraft hat also der obige Ausführungsplan für ein in der Praxis auftretendes Performanceproblem?

Führt man das SQL-Statement tatsächlich aus und ermittelt dann den tatsächlichen Ausführungsplan so zeigt sich, dass bei einer Belegung der Bindevariable T mit dem Datum "01/01/2000 00:00:00" tatsächlich ein FULL TABLE SCAN, wie vom EXPLAIN PLAN Kommando vorausgesagt, ausgeführt wird:

```
-------------------------------------------------------------------------------
| Id  | Operation         | Name        | E-Rows | Cost (%CPU)| A-Rows |   A-Time
-------------------------------------------------------------------------------
|*  1 |  TABLE ACCESS FULL| T_TEST_DATA |   1953K|   106   (3)|    310 |00:00:00.01
-------------------------------------------------------------------------------
```

Hat die Bindevariable T aber einen abweichenden Datumswert, ergibt sich ein Zugriff über den Index:

```
-------------------------------------------------------------------------------
| Id  | Operation                   | Name                       | Cost | A-Rows |…
-------------------------------------------------------------------------------
|   1 |  TABLE ACCESS BY INDEX ROWID| T_TEST_DATA                |     2|      0 |…
|*  2 |   INDEX RANGE SCAN          | T_TEST_DATA_DATE2DISTINCT  |     1|      0 |…
-------------------------------------------------------------------------------
```

Bei der Ausführung von SQL-Statements führt der Optimizer ein "Bind Variable Peeking" aus, d.h. der Optimizer bestimmt den Ausführungsplan in Abhängigkeit der Wertebelegung der Bindevariablen.

Diese "Bind Variable Peeking" wird, bei vorhandener Option "PEEKED_BINDS" im Parameter FORMAT der DBMS_XPLAN.DISPLAY* Funktionen, auch im Ausführungsplan dokumentiert:

```
...
Peeked Binds (identified by position):
--------------------------------------

   1 - (DATE): 01/01/00 00:00:00
...
```

1.2) EXPLAIN PLAN und Typen von Bindevariablen

Wir das SQL Statement

```
SELECT * FROM MUMBAI42.t_test_data
WHERE DATE_TWO_DISTINCT = :T;
```

an den Optimizer übergeben und mit ihm die Bindevariable T vom Typ DATE, so hat der Optimizer die Möglichkeit den zur Verfügung stehenden Index auf der Spalte DATE_TWO_DISTINCT zu nutzen und wird dies, unter anderem in Abhängigkeit von der Wertebelegung von T, auch tun.

Ist die Bindevariable allerdings als Datentyp TIMESTAMP definiert, wird der Optimizer die Nutzung des Index nicht mehr in Betracht ziehen, da sich Datentyp der indexierten Spalte (DATE) und der Bindevariable (TIMESTAMP) unterscheiden – es kommt zum sogenannten "Bind variable type mismatch".

Da EXPLAIN PLAN keine Bindevariablen auswertet, folgt dass EXPLAIN PLAN auch den "Bind variable type mismatch" nicht erkennt und sich dieses Problem damit auch eben nicht mit EXPLAIN PLAN untersuchen lässt. EXPLAIN PLAN würde im Beispiel den Zugriff über den Index anzeigen, bei der Ausführung würde aber ein FULL TABLE SCAN genutzt werden.

Dass ein „Bind variable type mismatch“ vorliegt ist entweder aus den Typdefinitionen von Bindevariablen in einem 10046 SQL Tracefile abzulesen oder mit einem wachen Auge der "Predicate Information"-Sektion eines mit DBMS_XPLAN angezeigten Ausführungsplans zu entnehmen:

```
Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter(INTERNAL_FUNCTION("DATE_TWO_DISTINCT")=:T)
```

Hier ist zu erkennen, dass über eine Oracle-interne Funktion eine Umwandlung der Werte in der Spalte DATE_TWO_DISTINCT, nämlich in den Typ der Bindevariablen, stattfindet.

1.3) Implementierung in Mumbai

In Mumbai existieren drei Möglichkeiten einen Ausführungsplan zu generieren (siehe Abb. 1):

- **Explain plan**

- **Parse, Execute and show plan**
  Das Statement wird geparst, ausgeführt, allerdings ohne auch nur einen Ergebnisdatensatz zu ermitteln. Hierbei wird ein Ausführungsplan generiert, der den Wert und Typ von Bindevariablen beachtet. Auch für langlaufenden SQL Statements kann so schnell ein exakter Ausführungsplan ermittelt werden.

- **Parse, Execute, Fetch all and Show plan**
  Das Statement wird geparst, ausgeführt und alle Datensätze gefetcht. Der gezeigte Ausführungsplan entspricht dem oben, aber zusätzlich wird die Ausführungsstatistik angezeigt, sodass auch ein Abgleich zwischen Abschätzungen (E-Spalten) des Optimizer und tatsächlich gemessenen Werten (A-Spalten) möglich ist:

```
…-------------------------------------------------------------------------
…| Starts | E-Rows |E-Bytes| Cost (%CPU)| A-Rows |   A-Time   | Buffers |
…-------------------------------------------------------------------------
…|      1 |   1000 | 20000 |   108   (5)|      0 |00:00:00.03 |     363 |
…-------------------------------------------------------------------------
```

  Die Erstellung des Ausführungsplans dauert mindestens so lange wie die Laufzeit des SQL-Statements ist, was bei Langläufern problematisch sein kann. Wenn auf Ausführungsstatistiken verzichtet werden kann, ist die obige Alternative „Parse, Execute and show plan“ ggf. sinnvoller.

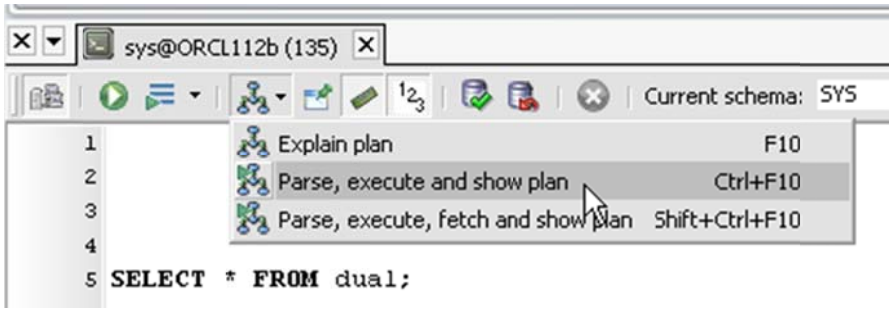
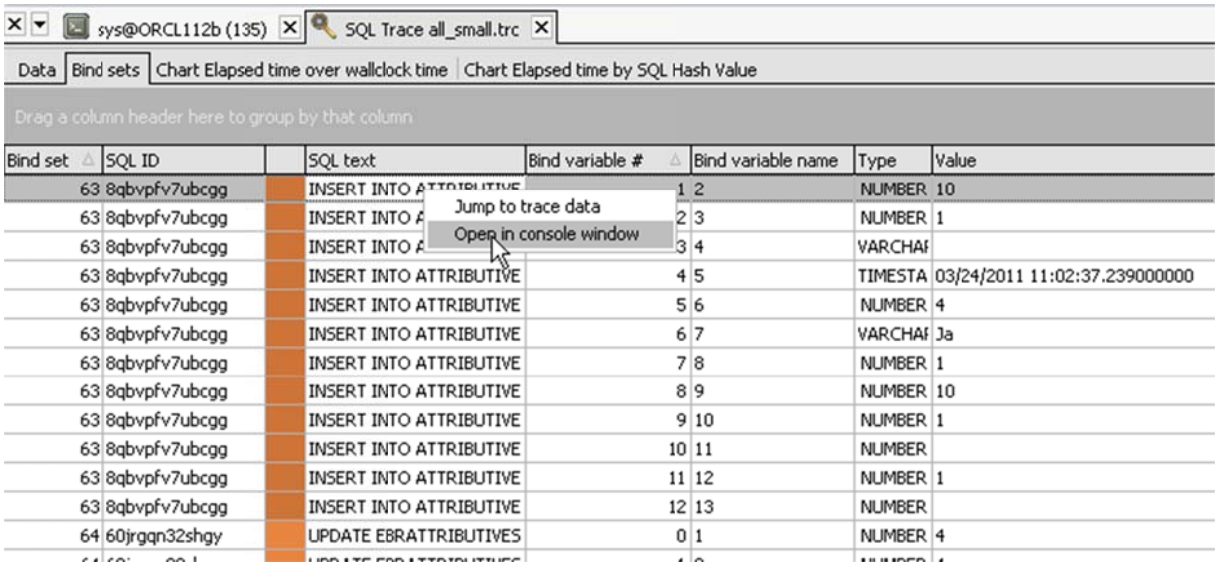


Abb. 1: Möglichkeiten zur Erstellung eines Ausführungsplans in Mumbai

Weiterhin besteht in Mumbai die Möglichkeit sich aus einem Satz von Bindevariablen in einem geladenen 10046 SQL Tracefile die einem SQL-Statement zugeordnet sind, automatisiert ein "Tuning set" zu erstellen umso das SQL Statement einschließlich Bindevariablen mit realistischem Datentyp weiter analysieren zu können (siehe Abb. 2).



Bind set	SQL ID		SQL text	Bind variable #	Bind variable name	Type	Value
63	8qbvpfv7ubcgg		INSERT INTO ATTRIBUTIVE	1	2	NUMBER	10
63	8qbvpfv7ubcgg		INSERT INTO A	2	3	NUMBER	1
63	8qbvpfv7ubcgg		INSERT INTO A	3	4	VARCHAI	
63	8qbvpfv7ubcgg		INSERT INTO ATTRIBUTIVE	4	5	TIMESTA	03/24/2011 11:02:37.239000000
63	8qbvpfv7ubcgg		INSERT INTO ATTRIBUTIVE	5	6	NUMBER	4
63	8qbvpfv7ubcgg		INSERT INTO ATTRIBUTIVE	6	7	VARCHAI	Ja
63	8qbvpfv7ubcgg		INSERT INTO ATTRIBUTIVE	7	8	NUMBER	1
63	8qbvpfv7ubcgg		INSERT INTO ATTRIBUTIVE	8	9	NUMBER	10
63	8qbvpfv7ubcgg		INSERT INTO ATTRIBUTIVE	9	10	NUMBER	1
63	8qbvpfv7ubcgg		INSERT INTO ATTRIBUTIVE	10	11	NUMBER	
63	8qbvpfv7ubcgg		INSERT INTO ATTRIBUTIVE	11	12	NUMBER	1
63	8qbvpfv7ubcgg		INSERT INTO ATTRIBUTIVE	12	13	NUMBER	
64	60jrgqn32shgy		UPDATE EBRATTRIBUTIVES	0	1	NUMBER	4

Abb. 2: Erstellung eines „Tuning sets“ auf Basis der Bindevariablen in einem 10046 SQL Tracefile in Mumbai

1.4) Zusammenfassung

Ausführungspläne für SQL Statements die Bindevariablen enthalten, sollten niemals mittels EXPLAIN PLAN ermitteln werden. Ein so ermittelter Ausführungsplan führt in die Irre und kann dazu führen, dass man ein Problem bekämpft, dass gar nicht existiert.

2.) Was passiert gerade auf der Datenbank?

Als Datenbankadministrator wird man regelmäßig mit der Aussage „Die Datenbank ist langsam.“ oder auch „Die Datenbank funktioniert nicht.“ von Endbenutzern oder Applikationsbetreuern konfrontiert. An dieser Stelle sollte man sicher erst einmal versuchen weitere Informationen zum tatsächlich vorliegenden Problem zu erfragen, aber manchmal bringt dies keine Eingrenzung des Problems oder aber eine solche Eingrenzung lässt sich nicht mit vertretbarem Aufwand erreichen.
So frustrierend es ist von der Idee einer Root-Cause-Analyse (vgl. dazu [3]) abzuweichen, lässt es sich in diesen Situationen nicht verhindern.
Ein Weg kann hier sein, erst einmal losgelöst vom vorliegenden Problem zu betrachten was zurzeit auf der Datenbank passiert.

Tools wie der Enterprise Manager von Oracle, TOAD oder Spotlight on Oracle, oft unterstützt von Active Workload Repositry (AWR) bzw. Active Session History (ASH) Funktionalität in der Datenbanksoftware, bieten hier einige Möglichkeiten. Der Einsatz dieser Tools erfordert aber auch immer dessen Verfügbarkeit, sowie eine entsprechende Lizensierung. So muss die AWR und ASH Funktionalität über das Oracle Diagnostic Pack lizensiert werden, was überhaupt nur für eine Enterprise Edition einer Datenbank möglich ist.

Aus Consultant-Sicht sind die Voraussetzungen Enterprise Edition und Diagnostic Pack nur bei wenigen Kunden erfüllt, was einem einheitlichen Vorgehensmodell im Wege steht. Daher soll im Folgenden kurz das von Tanel Poder entwickelte „Session Snapper“ PL/SQL-Skript [2] vorgestellt werden, welches für Standard und Enterprise Edition Datenbanken ab Version 9.2 eingesetzt werden kann.

2.1) Einführung zu Snapper

Plakativ beschrieben ist Snapper eine Ad-hoc Version von AWR und ASH für kurze Zeitintervalle ohne Lizensierungskosten und Installationsaufwand und ohne historischen Datenhaltung.

Beim Snapper Skript handelt es sich um einen anonymen PL/SQL Block, der zum Zeitpunkt des Ausrufes von der Datenbank geparst und ausgeführt wird, d.h. es müssen keine Objekte in der zu untersuchenden Datenbank angelegt werden.
Abgesehen von der Zugangsmöglichkeit zur Datenbank mit einem entsprechenden privilegierten Benutzer (z.B. dem Benutzer system), sind keine weiteren Voraussetzungen für den Einsatz zu erfüllen.

Wie erwähnt kann man die Funktionalität von Snapper mit Statspack, AWR und ASH vergleichen. Zum einen macht es zu Beginn und zum Ende eines gewünschten Zeitintervalls (üblicherweise einige Sekunden, ggf. Minuten) Snapshots von einigen V$ und X$ Views, berechnet die Vorher-Nachher-Differenzen für die einzelnen Werte in diesen Views und gibt die ermittelten Deltas aus.

Neben dieser mit AWR und Statspack vergleichbaren Funktionalität bildet Snapper noch die ASH Funktionalität des Diagnostic Packs nach. Hier erfolgt während des gewünschten Zeitintervalls wiederholt ein Sampling von Daten aus V$SESSION (speziell bezüglich Wait-Events), sodass am Ende des Messintervalls eine recht genaue Ausgabe der Aktivitäten auf der Datenbank möglich ist.

2.2) Nutzung von Snapper

Um Snapper nutzen zu können, öffnet man eine SQL*Plus Session zur Datenbank und ruft das snapper.sql Skript auf. Die zu übergebenden Parameter sind wie folgt dokumentiert:

```
snapper.sql
 <ash[1-3]|stats|all>[,out][,trace][,pagesize=X][,gather=[s][t][w][l][e][b][a]]>
 <seconds_in_snap> <snapshot_count> <sid(s)_to_snap>
```

Alle Details zu den Parametern sind im Skript selbst sowie unter [2] dokumentiert. Beispielhaft sind im Folgenden einige Aufrufe und Ausgaben dargestellt.

A.) Ausgabe der Deltas für 15 Sekunden aus V$SESSSTAT für die Session mit der SID 1305

```
SQL> @snapper stats,gather=s 15 1 1305
Sampling SID 1305 with interval 15 seconds, taking 1 snapshots...

-- Session Snapper v3.52 by Tanel Poder @ E2SN ( http://tech.e2sn.com )

--------------------------------------------------------------------------------
  SID, USERNAME  , TYPE, STATISTIC                      ,   DELTA, HDELTA/SEC
--------------------------------------------------------------------------------
 1305, APPUSER   , STAT, opened cursors cumulative      ,     412,      27.47
 1305, APPUSER   , STAT, user calls                     ,    1442,      96.13
 1305, APPUSER   , STAT, session logical reads          ,    6554,     436.93
 1305, APPUSER   , STAT, CPU used when call started     ,       6,         .4
 1305, APPUSER   , STAT, CPU used by this session       ,       6,         .4
 1305, APPUSER   , STAT, DB time                        ,      10,        .67
 1305, APPUSER   , STAT, user I/O wait time             ,       4,        .27
 1305, APPUSER   , STAT, physical read total IO requests,      33,        2.2
 1305, APPUSER   , STAT, physical read total bytes      ,  270336,     18.02k
 1305, APPUSER   , STAT, consistent gets                ,    6554,     436.93
 1305, APPUSER   , STAT, consistent gets from cache     ,    6554,     436.93
 1305, APPUSER   , STAT, consistent gets - examination  ,     875,      58.33
 1305, APPUSER   , STAT, physical reads                 ,      33,        2.2
 1305, APPUSER   , STAT, physical reads cache           ,      33,        2.2
 1305, APPUSER   , STAT, physical read IO requests      ,      33,        2.2
 1305, APPUSER   , STAT, physical read bytes            ,  270336,     18.02k
 1305, APPUSER   , STAT, free buffer requested          ,      33,        2.2
 1305, APPUSER   , STAT, dirty buffers inspected        ,       4,        .27
 …
 1305, APPUSER   , STAT, parse count (total)            ,     412,      27.47
 1305, APPUSER   , STAT, execute count                  ,     412,      27.47
 1305, APPUSER   , STAT, bytes sent via SQL*Net to clien,  290968,      19.4k
 1305, APPUSER   , STAT, bytes received via SQL*Net from,  196569,      13.1k
 1305, APPUSER   , STAT, SQL*Net roundtrips to/from clie,    1030,      68.67
--  End of Stats snap 1, end=2011-09-08 14:36:49, seconds=15
```

B.) Ausgabe der Deltas für 15 Sek. aus V$SESS_TIME_MODEL für die Session 1346

```
SQL> @snapper stats,gather=t 15 1 1346
Sampling SID 1346 with interval 15 seconds, taking 1 snapshots...

-- Session Snapper v3.52 by Tanel Poder @ E2SN ( http://tech.e2sn.com )

--------------------------------------------------------------------------------
  SID, USERNAME, TYPE, STATISTIC                   ,   DELTA, HDELTA/SEC,%TIME
--------------------------------------------------------------------------------
 1346, APPUSER , TIME, hard parse elapsed time     ,    4689,    312.6us,  .0%
 1346, APPUSER , TIME, repeated bind elapsed time  ,   10303,   686.87us,  .1%
 1346, APPUSER , TIME, parse time elapsed          ,    6932,   462.13us,  .0%
 1346, APPUSER , TIME, DB CPU                      , 1202539,    80.17ms, 8.0%
 1346, APPUSER , TIME, sql execute elapsed time    , 1130622,    75.37ms, 7.5%
 1346, APPUSER , TIME, DB time                     , 1202435,    80.16ms, 8.0%
 1346, APPUSER , TIME, sequence load elapsed time  ,    1362,     90.8us,  .0%
--  End of Stats snap 1, end=2011-09-08 14:42:55, seconds=15
```

C.) Ausgabe der gesampelten ASH Werte für 15 Sekunden für alle Sessions – hier erfolgt die Ausgabe eines prozentualen Aktivitätswertes gruppiert und summiert nach sql_id des aktiven SQL Statements, Event-Name und Event-Klasse

```
SQL> @snapper ash=sql_id+event+wait_class 15 1 all
Sampling SID all with interval 15 seconds, taking 1 snapshots...

-- Session Snapper v3.52 by Tanel Poder @ E2SN ( http://tech.e2sn.com )

-----------------------------------------------------------------------
Active% | SQL_ID          | EVENT                     | WAIT_CLASS
-----------------------------------------------------------------------
    42% | cjcc7hn6kpmar   | ON CPU                    | ON CPU
     9% | 47cxgqhqmckzq   | ON CPU                    | ON CPU
     3% |                 | ON CPU                    | ON CPU
     3% | 27qxbhb0pyyuv   | db file sequential read   | User I/O
     3% | bppsa8jcgrxg4   | ON CPU                    | ON CPU
     3% | 8gansjz25ca2w   | ON CPU                    | ON CPU
     1% | 44d44xa9623by   | ON CPU                    | ON CPU
     1% | djzryk2qczyuf   | db file sequential read   | User I/O
     1% | 05qnygwpukbsw   | db file sequential read   | User I/O
     1% | 39hcyjdpfvjwx   | db file sequential read   | User I/O

--  End of ASH snap 1, end=2011-09-08 14:46:47, seconds=15, samples_taken=77
```

Die letzte Zeile zeigt unter anderem, dass 77 Samples aus V$SESSION innerhalb des 15 Sekundenintervalls genommen wurden.

D.) Wie C.), jedoch zusätzliche Gruppierung und Summierung nach SID

```
SQL> @snapper ash=sql_id+sid+event+wait_class 15 1 all
Sampling SID all with interval 15 seconds, taking 1 snapshots...

-- Session Snapper v3.52 by Tanel Poder @ E2SN ( http://tech.e2sn.com )

-------------------------------------------------------------------------------
Active% | SQL_ID          |    SID | EVENT                     | WAIT_CLASS
-------------------------------------------------------------------------------
     3% | 94hhjvkh1wxp0   |   1331 | db file sequential read   | User I/O
     3% | 4m4b09rfbh8yh   |   1528 | db file sequential read   | User I/O
     3% | bppsa8jcgrxg4   |   1123 | ON CPU                    | ON CPU
     1% | 1zdpkttw600ru   |   1583 | ON CPU                    | ON CPU
     1% | adqbh8yju7hj3   |   1022 | ON CPU                    | ON CPU
     1% | 62r2vmgz66544   |   1396 | db file sequential read   | User I/O
     1% | 51r70arqhzf77   |   1583 | ON CPU                    | ON CPU
     1% | 31jwcpaqm7232   |   1143 | db file sequential read   | User I/O
     1% | 6qnptxvatt8sr   |   1188 | SQL*Net more data to clie | Network
     1% | 5dadd4txbw6yv   |   1500 | db file sequential read   | User I/O

--  End of ASH snap 1, end=2011-09-08 14:49:54, seconds=15, samples_taken=79
```

Diese vier beispielhaften Aufrufe zeigen bereits wie vielseitig Snapper in der Ad-Hoc Analyse von Performanceproblemen eingesetzt werden kann. Speziell der ASH Modus mit den verschiedenen Gruppierungsmöglichkeiten können hier sehr hilfreich sein.

2.4) Unterstützung von Snapper in Mumbai

Snapper lässt sich über einen graphischen Optionsdialog aus Mumbai heraus starten und Mumbai zeigt die gesammelten Daten kontinuierlich in tabellarischer Form sowie in Diagrammen an.

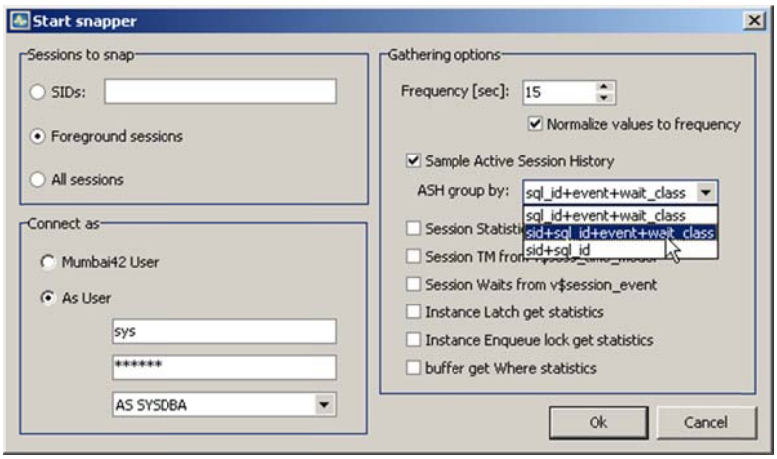


Abb. 3: Optionsdialog zum Starten von Snapper aus Mumbai

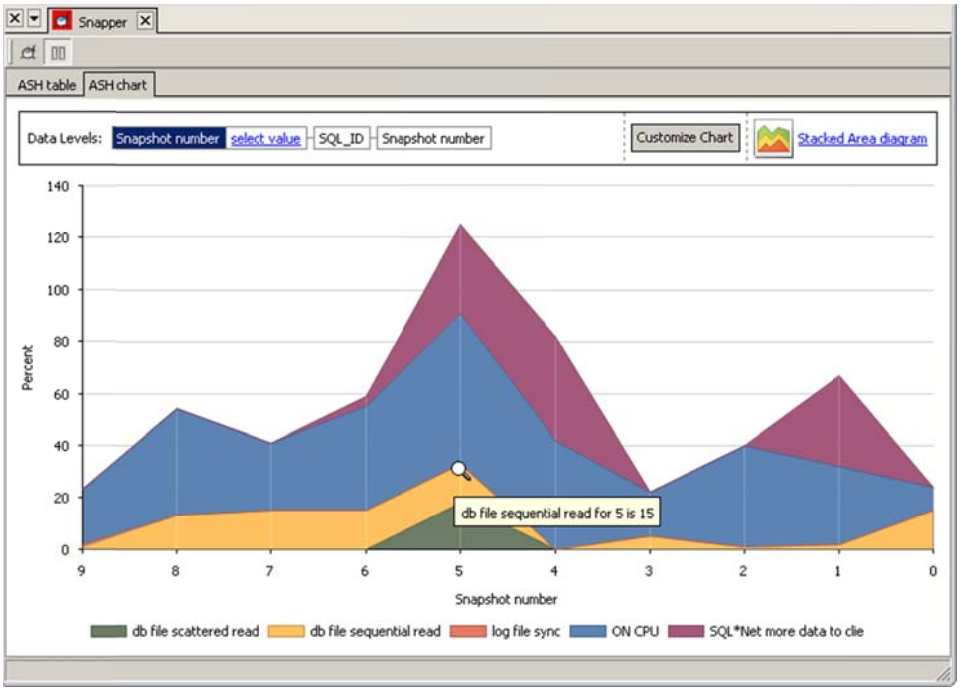


Abb. 4: Darstellung von durch Snapper gesammelter ASH Daten in Mumbai

Die graphische Darstellung in Mumbai mit der Möglichkeit Datenreihen zu filtern und sortieren, sowie über „Data Level“ Detailinformationen anzuzeigen die in der gruppierten Ansicht der Ergebnisse nicht sichtbar sind, eignet sich gut für einen schnellen Überblick über problematische Sessions bzw. SQL Statements. Für die weitergehende Performanceanalyse im Detail ist es aber oft sinnvoll oder notwendig Snapper manuell über eine SQL*Plus Session auszuführen, da sich nur so alle Möglichkeiten des Kommandozeilenaufrufs mit allen Parametern ergeben.

2.3) Zusammenfassung

Snapper eignet sich hervorragend um sich einen schnellen Überblick über die aktuelle Situation in einer Datenbank zu machen. Da es sich lizensierungsfrei und ohne Anlegen von Objekten in der Datenbank auf allen Oracle Datenbanken ab Version 9.2 nutzen lässt, ist es eine sehr gute, mächtige und allgemein verfügbare Alternative für entsprechende kommerzielle Lösungen.

3.) Was passierte in der Vergangenheit auf der Datenbank?

3.1) Problem der Durchschnittsbildung

Um sich rückwirkend einen Überblick über die Aktivität in einer Datenbank zu machen, bieten sich AWR (so vorhanden und lizensiert), sowie Statspack an. Beide sehr ähnlichen Tools ermöglichen die Erstellung von Reports für wählbare Intervalle. Wird beispielweise alle 30 Minuten ein AWR bzw. Statspack Snapshot erstellt, so ist dies auch das minimale Intervall für den sich ein entsprechender Report mit den von Oracle gelieferten Skripten (bzw. im Fall von AWR auch mit PL/SQL Packages) erstellen lässt. Nach oben ist als Intervall alles möglich, was noch mit entsprechenden AWR-bzw. Statspackdaten hinterlegt ist. Hier sind also Intervallen von mehrere Stunden, Tagen, Wochen, Monaten oder gar Jahren denkbar.

In wieweit sind nun „lange“ Intervalle für Auswertungen von AWR/Statspack Daten sinnvoll? Die in einem Report gelieferten Daten basieren immer auf einer Differenzbildung (Endwert-Startwert) eines Performancewertes aus der Datenbank - ggf. wird noch normalisiert in dem durch die Intervallzeit geteilt wird.
Es findet hier also eine Durchschnittswertbildung statt und um Cary Millsap zu zitieren, gilt: *You can't extrapolate detail from an aggregate.*[3] Betrachtet man also lange Zeiträume bei der Auswertung von AWR/Statspack-Daten so gehen Information verloren, die sich aus dem Durchschnittswert auch nicht wieder ableiten lassen.

Beispiel: Ein nur ein, zwei Mal ausgeführtes SQL-Statement, mit einer Ausführungszeit von je 30 Minuten, wird in einem Statspack-Report über einen Zeitraum von einer Woche (entspricht 10.080 Minuten) vermutlich gar nicht im Report erscheinen, da es im Vergleich zur Intervallzeit und meist auch im Vergleich zur Ausführungszeit andere SQL-Statements (regelmäßig laufende Maintenance- oder Batch-Jobs) vernachlässigt werden kann. Trotzdem kann eine Ausführungszeit von 30 Minuten natürlich für einen Anwender der darauf wartet ein Problem sein.

Um sich also rückwirkend einen Überblick über die Aktivitäten und Probleme in einer Datenbank ein Bild zu machen, reicht es nicht einen Report mit einem großen Intervall zu erstellen. Idealerweise müsste man für jeden Snapshot einen Report bis zum entsprechenden Snapshotvorgänger erstellen – um ein Wochenintervall mit Snapshots alle 30 Minuten zu analysieren, müssten also 7*24*2=336 Reports erstellt werden. Technisch keine große Hürde, aber leider müsste man alle 336 Reports auch noch analysieren…

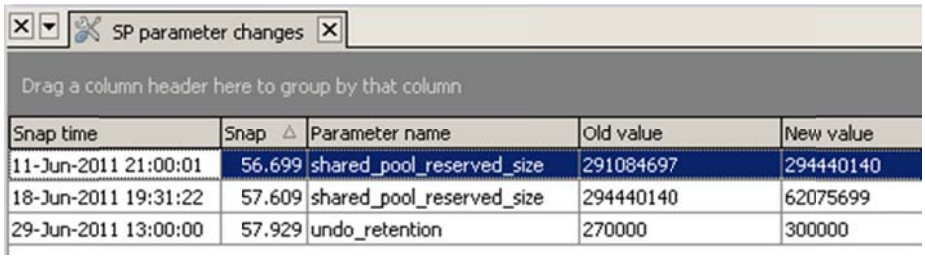
Es wird klar, dass textuelle Reports für einen schnellen Überblick über lange Zeiträume nicht zum Ziel führen. Wünschenswert wären graphische Darstellungen die über Datum und Zeit den Verlauf gewisser Werte der in Statspack/AWR-Reports dargestellten Ergebnisse anzeigen. In Teilen ist dies im Oracle Enterprise Manager mit AWR Daten möglich, allerdings nicht in einem Detailierungsgrad wie ihn die textuellen Reports liefern. Für Statspackdaten besteht die Möglichkeit der graphischen Darstellung gar nicht.

3.2) Auswertung von AWR/Statspack Daten in Mumbai

Mumbai unterstützt die Erstellung von tabellarischen Übersichten und graphischen Diagrammen über den Analysezeitraum für folgende Basisdaten:

- Parameteränderungen (nur tabellarisch) (Abb. 5)
- Alle erfassten Werte aus V$SYSSTAT über den „Instance Activity Report“
- Wait events und CPU-Zeit (Abb. 6)
- Average Active Session (AAS)
- Time Model Statistics
- OS Statistics
- Top SQL nach Elapsed time, CPU time, Buffer gets und Anzahl von Ausführungen (Abb. 7)

Die dargestellten Werte basieren nicht auf einer einfachen Differenzbildung zwischen End- und Start-Snapshot, sondern es werden für alle vorhandenen, aufeinander folgenden Snapshots im gewünschten Analysezeitraum die entsprechenden Deltas gebildet und graphisch über die Zeit dargestellt.



Snap time	Snap	Parameter name	Old value	New value
11-Jun-2011 21:00:01	56.699	shared_pool_reserved_size	291084697	294440140
18-Jun-2011 19:31:22	57.609	shared_pool_reserved_size	294440140	62075699
29-Jun-2011 13:00:00	57.929	undo_retention	270000	300000

Abb. 5: Auswertung der Parameteränderungen aus Statspackdaten

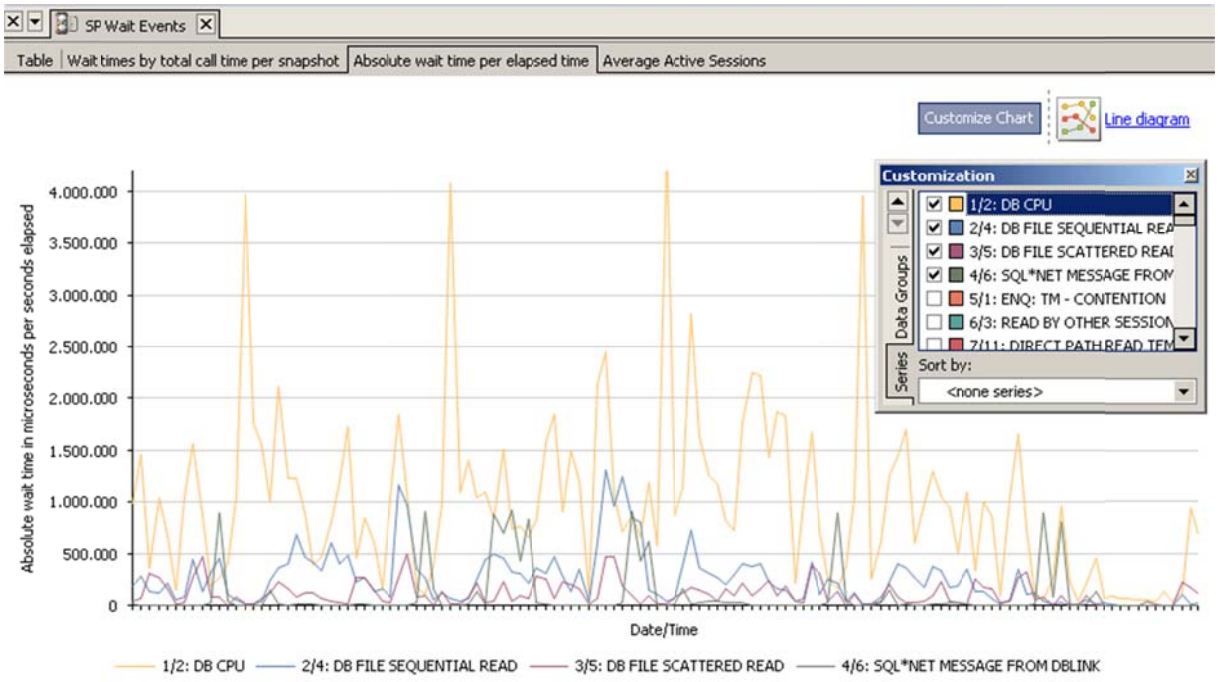


Abb. 6: Graphische Darstellung von Wait-Events aus Statspackdaten

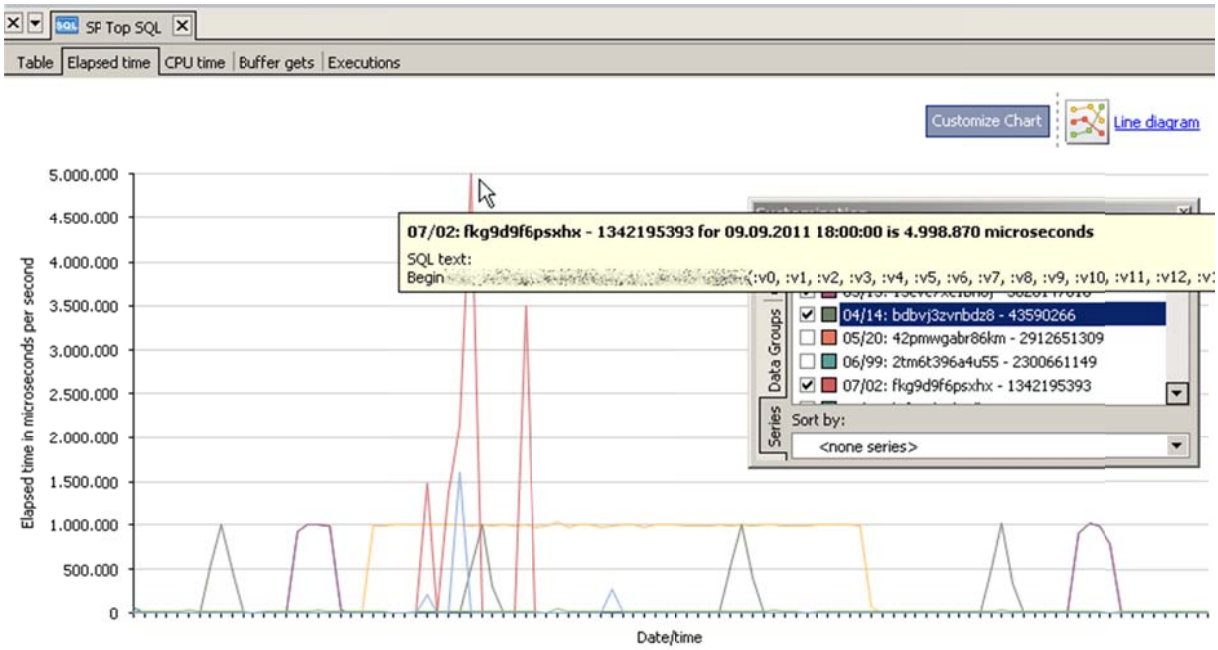


Abb. 7: Graphische Darstellung von Top SQL nach Elapsed time aus Statspackdaten

3.3) Zusammenfassung

Die Möglichkeit Statspack/AWR-Daten in Mumbai zu visualisieren, ohne dabei Detailinformationen zu verlieren, macht es zu einem sehr nützlichen Tool bei der rückwirkenden Analyse von Datenbankaktivität. Dabei ist es kein Ersatz für die ausführlichen, textuellen Statspack/AWR-Reports, sondern vielmehr lassen sich mit dieser Funktionalität Intervalle finden in denen die Generierung einzelner Statspack/AWR-Reports sinnvoll ist.

Verweise

[1] Marcus Mönnig's Oracle and Mumbai Blog - http://marcusmonnig.wordpress.com
[2] Tanel Poder’s Session Snapper - http://tech.e2sn.com/oracle-scripts-and-tools/session-snapper
[3] Cary Millsap, Jeff Holt – Optimizing Oracle Performance – ISBN: 059600527X

Kontaktadresse

Marcus Mönnig
Lichtblick AG
Zirkusweg 6
D-20359 Hamburg

Telefon: +49 (0) 40 6360-1305
E-Mail mm@marcusmoennig.de
Internet: http://marcusmonnig.wordpress.com/