**Learn Oracle Performance Tuning with Real-Time scenario by Industry Expert in 30 Day**

* **AWR ,ASH and ADDM Analysis**
* **Oracle Performance Tuning Technique**
* **Advance Database Performance Tools**

**( RDA and OSWATCHER)**

* **Various OS Commands For Identifying Bottleneck**
* **Index Rebuild Concept and Test Case**

**3)AWR Report:**

Oracle have provided many performance gathering and reporting tools over the years. Originally the UTLBSTAT/UTLESTAT scripts were used to monitor performance metrics. Oracle8i introduced the Statspack functionality which Oracle9i extended. In Oracle 10g statspack has evolved into the Automatic Workload Repository (AWR).

## AWR Features

The AWR is used to collect performance statistics including:

* Wait events used to identify performance problems.
* Time model statistics indicating the amount of DB time associated with a process from the V$SESS\_TIME\_MODEL and V$SYS\_TIME\_MODEL views.
* Active Session History (ASH) statistics from the V$ACTIVE\_SESSION\_HISTORY view.
* Some system and session statistics from the V$SYSSTAT and V$SESSTAT views.
* Object usage statistics.
* Resource intensive SQL statements.

## Workload Repository Views

The following workload repository views are available:

* V$ACTIVE\_SESSION\_HISTORY - Displays the active session history (ASH) sampled every second.
* V$METRIC - Displays metric information.
* V$METRICNAME - Displays the metrics associated with each metric group.
* V$METRIC\_HISTORY - Displays historical metrics.
* V$METRICGROUP - Displays all metrics groups.
* DBA\_HIST\_ACTIVE\_SESS\_HISTORY - Displays the history contents of the active session history.
* DBA\_HIST\_BASELINE - Displays baseline information.
* DBA\_HIST\_DATABASE\_INSTANCE - Displays database environment information.
* DBA\_HIST\_SNAPSHOT - Displays snapshot information.
* DBA\_HIST\_SQL\_PLAN - Displays SQL execution plans.
* DBA\_HIST\_WR\_CONTROL - Displays AWR settings.

## Workload Repository Reports

Oracle provide two scripts to produce workload repository reports (awrrpt.sql and awrrpti.sql). They are similar in format to the statspack reports and give the option of HTML or plain text formats. The two reports give essential the same output but the awrrpti.sql allows you to select a single instance. The reports can be generated as follows.

AWR report can be generated upon setting the parameter is called statistics\_level=typical or all, If set to basic it will be disabled. We can get the value like below.

SQL> show parameter statistics\_l

NAME TYPE VALUE

------------------------------------ ----------- ------------------------------

statistics\_level string TYPICAL

SQL>

1. **Snapshots**

Snapshots are sets of historical data for specific time periods that are used for performance comparisons by ADDM. By default, Oracle Database automatically generates snapshots of the performance data once every hour and retains the statistics in the workload repository for 8 days. You can also manually create snapshots, but this is usually not necessary. The data in the snapshot interval is then analyzed by the Automatic Database Diagnostic Monitor (ADDM)

1. **Managing Snapshots:-**

By default, Oracle Database generates snapshots once every hour, and retains the statistics in the workload repository for 8 days. When necessary, you can use DBMS\_WORKLOAD\_REPOSITORY procedures to manually create, drop, and modify the snapshots. To invoke these procedures, a user must be granted the DBA role.

* 1. Creating Snapshots :- You can manually create snapshots with the CREATE\_SNAPSHOT procedure to capture statistics at times different than those of the automatically generated snapshots. For example:

BEGIN

DBMS\_WORKLOAD\_REPOSITORY.CREATE\_SNAPSHOT ();

END;

/

* 1. Dropping Snapshots:- You can drop a range of snapshots using the DROP\_SNAPSHOT\_RANGE procedure. To view a list of the snapshot IDs along with database IDs, check the DBA\_HIST\_SNAPSHOT view. For example, you can drop the following range of snapshots:

BEGIN

DBMS\_WORKLOAD\_REPOSITORY.DROP\_SNAPSHOT\_RANGE (low\_snap\_id => ,

high\_snap\_id => , dbid =>);

END;

/

* 1. Modifying Snapshot Settings:- You can adjust the interval, retention, and captured Top SQL of snapshot generation for a specified database ID, but note that this can affect the precision of the Oracle Database diagnostic tools. The INTERVAL setting affects how often the database automatically generates snapshots. The RETENTION setting affects how long the database stores snapshots in the workload repository. The TOPNSQL setting affects the number of Top SQL to flush for each SQL criteria (Elapsed Time, CPU Time, Parse Calls, sharable Memory, and Version Count). The value for this setting is not affected by the statistics/flush level and will override the system default behavior for the AWR SQL collection. It is possible to set the value for this setting to MAXIMUM to capture the complete set of SQL in the shared SQL area, though by doing so (or by setting the value to a very high number) may lead to possible space and performance issues because there will more data to collect and store. To adjust the settings, use the MODIFY\_SNAPSHOT\_SETTINGS procedure. For example:

BEGIN

DBMS\_WORKLOAD\_REPOSITORY.MODIFY\_SNAPSHOT\_SETTINGS( retention => 43200,

interval => 30, topnsql => 100, dbid => 3310949047);

END;

/

1. **Generate AWR report:-**

The awrrpt.sql SQL script generates an HTML or text report that displays statistics for a range of snapshot IDs. Below is the example.

Generating an Oracle RAC AWR Report:-

The awrgrpt.sql SQL script generates an HTML or text report that displays statistics for a range of snapshot IDs using the current database identifier and all available database instances in an Oracle Real Application Clusters (Oracle RAC) environment.

Generate a report for specific Instance.

The awrrpti.sql SQL script generates an HTML or text report that displays statistics for a range of snapshot IDs using a specific database and instance. This script enables you to specify a database identifier and instance for which the AWR report will be generated.

***Details AWR Analysis Concept***

The AWR report contains a significant amount of information which helps to focus on certain areas to get started.

AWR sample information at particular times. The AWR default sample is every 60 minutes.

By default, Oracle Database automatically generates snapshots once every hour.

**Checklist**

1) Review Overall picture from AWR header information

2) Check Host and Instance CPU to determine the proportion of CPU usage by this instance

3) Check the Load profile to use later in the context of the top waits

4 )Examine Top 5 Timed Events for highest resource users

**Checklist Detail**

This checklist steps through the recommended areas to investigate when presented with an AWR report. It does not assume

that you have any information other than the database is performing slowly and what is contained in the AWR output.

* ***Review Overall picture from AWR header information***

The header section contains useful information that can help set the context of the report you are looking at.

For example, the report contains a number of sections that quote specific counts of various statistics.

Without a timescale, these numbers are meaningless.

* + Release  
    Depending on the problem, the database version may be important. If the version is old or is not the latest patchset release then the most up to date fixes may not be applied which has the potential to open the database up to issues.
  + RAC  
    If the database is running in a Real Application Cluster configuration then you may need to look at information from the other instances to get a full picture of the database performance
  + Platform  
    There may be platform specific issues that have a bearing on the system
  + CPUs/Cores  
    In a multi-processor environment, the "wall clock" time is not necessarily a good indicator of how much work the database can do since multiple operations can be pursued simultaneously. You can use cores for an indication of how much CPU work can likely be done at once.
  + Snap Time  
    The Snap time shows the times for the starting and ending snapshots for the report period. Does this cover the time of problem that is being encountered?
  + Elapsed time  
    The elapsed time indicates the duration of the report between the 2 selected snapshots. Any other duration figures can be compared back to this. When looking at this figure, is the duration reasonable? If the duration is too short then important information may be missed. If it is too long then findings may be diluted. A 30-60 minute reporting period is usually recommended. In terms of AWR snapshots, as much as possible snapshots should be minimum 10 minutes, maximum 30 minutes.
  + DB time  
    The DB Time is the time spent in the database for the period of the report. If this is significantly higher than the Elapsed time then this is a good indicator of a heavily loaded system. Remember that on a multi-processor system, you might expect the DB Time to be able to exceed the elapsed time. Additionally, the db time includes the time waiting for the CPU to become available, so this number can be higher than the Elapsed time X Cores.   
    In the example above, the numbers say that the database worked for 2193 minutes in 15 minutes of elapsed time. Whether that is an indication of a problem depends on the capacity and concurrency capabilities of the system. Looking at the numbers, 2193:15 is a ratio of 146:1, so, in this case, if they had significantly less than 146 cpus it is likely that there is some overloading issues.  Remember that the user perception is also a significant factor in whether there is a "performance issue" - if the system delivers what the users want then there might not be a problem!
  + Sessions  
    You can use the sessions information along with the DB time to give an average amount of DB time per session. Are there a large number or a small number of connections?
* ***Check Host and Instance CPU to determine the proportion of CPU usage by this instance***

Another important area to look at before going to the detail of the top wait events is the Host and Instance CPU sections.  
These provide information regarding how much load there is on the underlying operating system and also how much of it is attributable to the instance in the AWR report. If the system is heavily loaded, then the performance of the database itself may be affected by the external contention. In these cases, look to see how much of the total CPU usage is being caused by this instance. In this case, 92.4% of the Total CPU can be attributed to the instance, which would tend to indicate that improving the instance performance is likely to improve the overall performance. If the instance was only responsible for a small proportion of the overall CPU, it may be that the problem lies elsewhere.

* ***Check the Load profile to use later in the context of the top waits***

The load profile section can provide you with a more detailed impression of where the database is loaded. Information is provided "Per Second" and "Per Transaction" for most statistics and also "Per Exec" and "Per Call" for DB Time and CPU.   
  
Suggested interpretations:

* + DB CPU(s)  
    The DB CPU(s) figure shows the amount of the CPU being used by the database. You can use this alongside the actual number of cores to give you an idea of how much of the total available machine CPU is actually being used by this instance.
  + DB Time(s) Here the "Per Second" information gives you another version of the total DB time used, just in this case expressed as every second as opposed to the full elapsed period.

Other statistics should be looked at within the context of the overall elapsed time and also in the context of the top waits, once you have looked at these later. For example:

* + Top events indicate library cache or cursor contention  
    In this case it would be sensible to look at the load in terms of Parse and Hard Parse statistics. The number of parses per execution could also be a relevant indicator
  + Top events are related to reading of blocks  
    In this case, do we see mainly physical or logical reads? If it is physical then are the explain plans for top queries such as to encourage more logical reads?

At this point you may also want to look at the Instance Efficiency Percentages to see if these bear out the findings from the above:  
  
  
Looking at these in the context of a specific wait is far more beneficial than attempting to reach 100%. If the bottleneck is elsewhere, attempting to change individual statistics will have little or no impact on the overall system. For example, in the Instance Efficiency Percentages above, the "Buffer Hit %" is 99.88%. If there is no contention for buffers and no waits for buffers, then what is the benefit in making changes to try to improve this number?    
  
You should also look at the numbers in the context of the other numbers. For example, in the case above, let us say that there is a performance issue and the top timed events showed that CPU usage was a significant resource. Looking at the "Parse CPU to Parse Elapsd %" alone, this says that 26.87% of the total parse time is CPU and maybe you would prefer a lower percentage (although 26% seems quite reasonable).  Since the:  "% Non-Parse CPU" is 98.07% this means that only 1.03% of the total CPU usage is parsing, so even if you reduced that 26.87% to the impossible value of zero then you would only gain 1% extra CPU overall. It is likely that you would need to look elsewhere for the cause of your CPU resource issue.

* ***Examine Top 5 Timed Events for highest resource users***

Once you have looked at the background information, the Top 5 Timed Events section is the place to start in order to tell what is taking up the largest proportion of the database time. Based upon the general feeling for the system, the top resource users are put in context and can be investigated to determine a root cause. This topic is covered in more detail in the following article:

### Interpretation

This document provides guidance on some background information to bear in mind when examining the detail of this section that can help by framing the context of the problem.  
  
Since we are looking at a performance issue, our primary concern is what the database is waiting for.  
When processes wait, they are being prevented from doing an activity because of some other factor. High waits provide the highest benefit when wait times are reduced and as such are a good focus.  
The Top Wait information provides such information and allows us to focus on the main problem areas without wasting time investigating areas that are not causing significant delay.

### Top 5 Timed Events

As mentioned, the Top waits section is the most important single section in the whole report being as it quantifies and allows comparison of the primary diagnostic: what each session is waiting for. An example output is provided below:

Top 5 Timed Events Avg %Total

~~~~~~~~~~~~~~~~~~ wait Call

Event Waits Time (s) (ms) Time Wait Class

------------------------------ ------------ ----------- ------ ------ ----------

db file scattered read 10,152,564 81,327 8 29.6 User I/O

db file sequential read 10,327,231 75,878 7 27.6 User I/O

CPU time 56,207 20.5

read by other session 4,397,330 33,455 8 12.2 User I/O

PX Deq Credit: send blkd 31,398 26,576 846 9.7 Other

-------------------------------------------------------------

The Top 5 Waits section reports on a number of useful topics related to Events. It records the number of waits encountered in the period and the total time spent waiting together with the average time waited for each event. The section is ordered by the %age of the total call time that each Event is responsible for.  
  
Dependent on what is seen in this section, other report sections may need to be referenced in order to quantify or check the findings. For example, the wait count for a particular event needs to be assessed based upon the duration of the reporting period and also the number of users on the database at the time; 10 Million waits in 10 minutes is far more significant than 10 Million in 10 hours, or if shared among 10 users as opposed to 10,000.  
  
In this example report, almost 60% of the time is spent waiting for I/O related reads.

* + Event 'db file scattered read ' is typically used when fetching blocks for a full table scan index fast full scan and performs multi-block IO.
  + Event 'db file sequential read'  is a single block read and is typically engaged for any activity where  multi-block IO is unavailable (for example index reads).

Another 20% of the time is spent waiting for or using CPU time. High CPU usage is often a symptom of poorly tuned SQL (or at least SQL which has potential to take less resource) of which excessive I/O can also be a symptom. More on CPU usage follows later.  
  
Based on this, we would investigate whether these waits indicate a problem or not. If so, resolve the problem, if not, move on to the next wait to determine if that is a potential cause.  
  
There are 2 main reasons why I/O related waits are going to be top of the waits:

* + The database is doing lots of reads
  + The individual reads are slow

The Top 5 events show us information that helps us here :

* + Is the database doing lots of reads?:  
    The section shows > 10 Million reads for each of these events in the period.  
    Whether this is a lot depends on whether the report duration is 1 hour or 1 minute.  
    Check the report duration to asses this.  
    If the reads do seem excessive, then why would the database do a lot of reads?  
    The database only reads data because the execution of SQL statements has instructed it to do so. To investigate further refer to the [SQL Statistics](https://support.oracle.com/epmos/faces/DocumentDisplay?_afrLoop=123610087304256&parent=DOCUMENT&sourceId=1628089.1&id=1359094.1&_afrWindowMode=0&_adf.ctrl-state=pvdrzedfb_243#SQLStats) Section.
  + Are the individual reads slow?  
    The section shows waits of <=8 ms for the 2 I/O related events.   
    Whether this is fast or slow is dependent on the hardware underlying the I/O subsystem, but typically anything under 20 ms is acceptable.  
      
    If the I/O was slow, then you can get further information from the 'Tablespace IO Stats ' section:
  + Tablespace IO Stats DB/Inst: VMWREP/VMWREP Snaps: 1-15
  + -> ordered by IOs (Reads + Writes) desc
  + Tablespace
  + ------------------------------
  + Av Av Av Av Buffer Av Buf
    - * Reads Reads/s Rd(ms) Blks/Rd Writes Writes/s Waits Wt(ms)
  + -------------- ------- ------ ------- ------------ -------- ---------- ------
  + TS\_TX\_DATA
  + 14,246,367 283 7.6 4.6 145,263,880 2,883 3,844,161 8.3
  + USER
  + 204,834 4 10.7 1.0 17,849,021 354 15,249 9.8
  + UNDOTS1
  + 19,725 0 3.0 1.0 10,064,086 200 1,964 4.9
  + AE\_TS
  + 4,287,567 85 5.4 6.7 932 0 465,793 3.7
  + TEMP
  + 2,022,883 40 0.0 5.8 878,049 17 0 0.0
  + UNDOTS3
  + 1,310,493 26 4.6 1.0 941,675 19 43 0.0
  + TS\_TX\_IDX
  + 1,884,478 37 7.3 1.0 23,695 0 73,703 8.3
  + SYSAUX
  + 346,094 7 5.6 3.9 112,744 2 0 0.0
  + SYSTEM

101,771 2 7.9 3.5 25,098 0 653 2.7

Specifically, look for the timing under Rd(ms).  If it is higher than 20 milliseconds per read and reads are high, then you may want to start investigating a potential I/O bottleneck from the os.

**NOTE:** You should ignore relatively idle tablespaces/files as you can get high values due to disk spin-up etc. which are not relevant. If you have an issue with 10 million reads being slow it is unlikely that a tablespace/file with 10 reads has caused the problem!

Although high waits for 'db file scattered read' and 'db file sequential read' can be I/O related, it is actually more common to find that these waits are relatively 'normal' based on the SQL that the database is being asked to run. In fact, on a well tuned database, you would want these events to be top of the waits, since that would mean that no 'problem' events were there instead!  
  
The trick is being able to assess whether the high waits is indicative of some SQL statements are not using optimal paths (as mentioned earlier) or otherwise.  If there are high waits for 'db file scattered read', then SQL may not be using optimal access paths and so are tending to do Full Table Scans as opposed to  indexes (or there may be missing indexes or not optimal indexes).  Furthermore, high waits for 'db file sequential read' may indicate SQL statements are using unselective indexes and there for reading more index blocks than necessary or using the wrong indexes.  So these waits may point to poor execution plans for SQL(s).    
  
In either case, the next step would be to check the top resource consuming SQL(s) from the AWR report to determine whether these look excessive or whether improvements can be made.  
  
  
As mentioned, 20% of the time is spent waiting for or using CPU time. This should also be looked at when looking at the SQL Statistics.

Remember that the next step to take following the Top 5 Waits is dependent upon the findings within that section. In the example above, 3 of the waits point towards potentially Sub-optimal SQL so that should be the section investigated next.  
  
Equally, if you do not see any latch waits, then latches are not causing a significant problem on your instance and so you do not need to investigate latch waits further.  
  
Generally, if the database is slow, and the Top 5 timed events include "CPU" and "db file sequential read" and "db file scattered read" in any order, then it is usually worth jumping to the Top SQL (by logical and physical reads) section of an AWR report and calling the SQL Tuning Advisor on them (or tune them manually) just to make sure that they are running efficiently.

### SQL Statistics

AWR Reports show a number of different SQL statistics:   
The different SQL statistic sub sections should be examined based upon the Top Wait events seen in the Top 5 Section.    
  
In our example, we saw top waits as 'db file scattered read' , 'db file sequential read' and CPU. For these, we are most interested in  SQL ordered by CPU Time, Gets and Reads.  These sections actually duplicate some information adding other specifics as appropriate to the topic.   
  
Often looking at 'SQL ordered by gets' is a convenient stating point as statements with high buffer gets are usually good candidates for tuning :

SQL ordered by Gets

-> Resources reported for PL/SQL code includes the resources used by all SQL

statements called by the code.

-> Total Buffer Gets: 4,745,943,815

-> Captured SQL account for 122.2% of Total

Gets CPU Elapsed

Buffer Gets Executions per Exec %Total Time (s) Time (s) SQL Id

-------------- ------------ ------------ ------ -------- --------- -------------

1,228,753,877 168 7,314,011.2 25.9 8022.46 8404.73 5t1y1nvmwp2

SELECT ADDRESSID",CURRENT$."ADDRESSTYPEID",CURRENT$URRENT$."ADDRESS3",

CURRENT$."CITY",CURRENT$."ZIP",CURRENT$."STATE",CURRENT$."PHONECOUNTRYCODE",

CURRENT$."PHONENUMBER",CURRENT$."PHONEEXTENSION",CURRENT$."FAXCOU

1,039,875,759 62,959,363 16.5 21.9 5320.27 5618.96 grr4mg7ms81

Module: DBMS\_SCHEDULER

INSERT INTO "ADDRESS\_RDONLY" ("ADDRESSID","ADDRESSTYPEID","CUSTOMERID","

ADDRESS1","ADDRESS2","ADDRESS3","CITY","ZIP","STATE","PHONECOUNTRYCODE","PHONENU

854,035,223 168 5,083,543.0 18.0 5713.50 7458.95 4at7cbx8hnz

SELECT "CUSTOMERID",CURRENT$."ISACTIVE",CURRENT$."FIRSTNAME",CURRENT$."LASTNAME",CU<

RRENT$."ORGANIZATION",CURRENT$."DATEREGISTERED",CURRENT$."CUSTOMERSTATUSID",CURR

ENT$."LASTMODIFIEDDATE",CURRENT$."SOURCE",CURRENT$."EMPLOYEEDEPT",CURRENT$.

Tuning can either be performed either manually or by calling the SQL Tuning Advisor on them:

#### *Analysis:*

* + -> Total Buffer Gets: 4,745,943,815  
    On the assumption that this is an hour long report, this is a significant number of gets and as such this confirms that it is worth investigating the top SQL statements to make sure they are taking optimal paths.
  + Individual Buffer Gets  
    The buffer gets for the individual statements shown are very high with the lowest being 850 Million. These 3 statements actually point towards 2 different reasons for the large number of buffers:
    - Excessive Buffer Gets/Execution  
      SQL\_IDs '5t1y1nvmwp2' and '4at7cbx8hnz' are only executed 168 times, but each execution reads over 5 Million buffers. This SQL statement is a prime candidate for tuning since the number of buffers read in each execution is so high.
    - Excessive Executions  
      On the other hand SQL\_ID 'grr4mg7ms81' only reads 16 buffers for each execution. Tuning the individual statement may not be able to reduce that significantly. However, the issue with this statement is caused by the number of times it is executed - 65 Million.   
      Changing the way in which the statement is called is likely to have the largest impact here - it is likely that the statement is called in a loop, once per record, if it could be called so as to process multiple records at once then there is potential for significant economies of scale.

Remember that these numbers may be 'normal' for this environment (since some are very busy).  By comparing this report against a baseline, you can see whether these SQL statements also read this much data when the database performs well. If they do then they are not the cause of the issue and can be ignored (although there may be benefit generally in improving them).

#### *Other SQL Statistic Sections*

As mentioned previously, there are a number of different report sections that help for specific causes. If you do not have the particular cause, then there is likely to be little benefit in looking at these. The following section outlines some potential causes and uses:

### Load Profile

Dependent on the waits, the load profile section either provides useful general background information or specific details related to potential issues.

Load Profile

~~~~~~~~~~~~ Per Second Per Transaction

--------------- ---------------

Redo size: 4,585,414.80 3,165,883.14

Logical reads: 94,185.63 65,028.07

Block changes: 40,028.57 27,636.71

Physical reads: 2,206.12 1,523.16

Physical writes: 3,939.97 2,720.25

User calls: 50.08 34.58

Parses: 26.96 18.61

Hard parses: 1.49 1.03

Sorts: 18.36 12.68

Logons: 0.13 0.09

Executes: 4,925.89 3,400.96

Transactions: 1.45

% Blocks changed per Read: 42.50 Recursive Call %: 99.19

Rollback per transaction %: 59.69 Rows per Sort: 1922.64

In the example, the waits section shows potential for issues with the execution of SQL so the load profile can be checked for details in this area, although it is not the primary source of such information.   
  
If you were looking at the AWR report for general tuning, you might pick up that the load section shows relatively high redo activity with high physical writes. There are more writes than reads on this load with 42% block changes.    
  
Furthermore, there is less hard parsing compared the soft parses.   
If there was a mutex wait as top wait such as 'library cache: mutex X', then statistics such as the overall parse rate would be more relevant.    
  
Again, comparing to a baseline will provide the best information, for example, checking to see if the load has changed by comparing redo size, users calls, and parsing.

### Instance Efficiency

Again, instance efficiency stats are more use for general tuning as opposed to addressing specific issues (unless waits point at these).

Instance Efficiency Percentages (Target 100%)

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Buffer Nowait %: 99.91 Redo NoWait %: 100.00

Buffer Hit %: 98.14 In-memory Sort %: 99.98

Library Hit %: 99.91 Soft Parse %: 94.48

Execute to Parse %: 99.45 Latch Hit %: 99.97

Parse CPU to Parse Elapsd %: 71.23 % Non-Parse CPU: 99.00

The most important Statistic presented here from the point of view of our example is the '% Non-Parse CPU' because this indicates that almost all the CPU  time that we see in the Top Waits section is attributable to Execution and not parse, which means that tuning SQL may help to improve this.  
  
If we were tuning, then 94.48% soft parse rate would show a small proportion of hard parsing which is desirable.  The high execute to parse % indicates good usage of cursors.  Generally, we want the statistics here close to 100%, but remember that a few percent may not be relevant dependent on the application.  For example, in a data warehouse environment, hard parsing may be higher due to usage of materialized views and, or histograms.  So again comparing to baseline report when performance was good is important.

### Latch Activity

In the example, we are not seeing significant waits for latches so this section could be ignored.  
  
However, if latch waits were significant, then we would be looking for high latch sleeps under Latch Sleep Breakdown for latch free waits:

Latch Sleep Breakdown

\* ordered by misses desc

Latch Name

----------------------------------------

Get Requests Misses Sleeps Spin Gets Sleep1 Sleep2 Sleep3

-------------- ----------- ----------- ---------- -------- -------- --------

cache buffers chains

2,881,936,948 3,070,271 41,336 3,031,456 0 0 0

row cache objects

941,375,571 1,215,395 852 1,214,606 0 0 0

object queue header operation

763,607,977 949,376 30,484 919,782 0 0 0

cache buffers lru chain

376,874,990 705,162 3,192 702,090 0 0 0

Here the top latch is cache buffers chains. Cache Buffers Chains latches protect the buffers in the buffer cache that hold data that we have retrieved from disk. This is a perfectly normal latch to see when data is being read. When this becomes stressed, the sleeps figure tends to rise as sessions start to wait to get the buffers they require. Contention can be caused by poorly tuned SQL reading the same buffers.  
  
In our example, although the gets are high at 2.8 billion buffer gets, the sleeps at 41,336 is low.  Average number of sleeps per miss ratio (Avg Slps/Miss) is low. The reason for this is that the server is able to deal with this volume of data and so there is no significant contention on Cache Buffers Chains latches at this point.  
  
For other latch free waits, review the following Document to identify what type of latches to investigate:

### Notable timed and wait events:

#### *CPU time events*

Just because CPU comes as top timed event in AWR may not indicate a problem.  However, if performance is slow with high CPU usage, then start investigating the wait.  First, check to see if a sql is taking most CPU under SQL ordered by CPU Time in AWR:

SQL ordered by CPU Time

-> Resources reported for PL/SQL code includes the resources used by all SQL

statements called by the code.

-> % Total is the CPU Time divided into the Total CPU Time times 100

-> Total CPU Time (s): 56,207

-> Captured SQL account for 114.6% of Total

CPU Elapsed CPU per % Total

Time (s) Time (s) Executions Exec (s) % Total DB Time SQL Id

---------- ---------- ------------ ----------- ------- ------- -------------

20,349 24,884    168     121.12   36.2    9.1 7bbhgqykv3cm9

Module: DBMS\_SCHEDULER

DECLARE job BINARY\_INTEGER := :job; next\_date TIMESTAMP WITH TIME ZONE := :myda

te; broken BOOLEAN := FALSE; job\_name VARCHAR2(30) := :job\_name; job\_subname

VARCHAR2(30) := :job\_subname; job\_owner VARCHAR2(30) := :job\_owner; job\_start

TIMESTAMP WITH TIME ZONE := :job\_start; job\_scheduled\_start TIMESTAMP WITH TIME

#### *Analysis:*

* + -> Total CPU Time (s): 56,207  
    This represents 15 minutes of CPU time in total. Whether this is significant depends on the report duration.
  + The top CPU using SQL uses 20,349 second (around 5 minutes),
  + Total DB of time this represents is 9.1%.
  + Executions is 168 - being as this execution count is the same as 2 of the 3 SQLs identified earlier, these may be related and this task may well be the scheduling job that runs the SQLs.

#### *Actions:*

Once you have identified the SQL statements that are using the highest CPU, investigate the reason for this usage.

* + Look at the number of executions and see whether that is appropriate for this statement. Excessive executions might indicate that the statement is being called too frequently and it might be possible to execute it for a group of rows rather than row by row (i.e. execute it in a batch).
  + Is the amount of CPU per execution excessive - this might indicate that the statement itself is inefficient.
  + Additionally, look at the other SQL Statistics in the AWR report to see if the SQLID(s) in question show excessive values for any of those, then deal with the statement appropriately.
* ***Details ADDM Analysis Concept***

Real –Time Analysis

**Important which need to address from AWR report.**

1) Load Profile

2) Instance Efficiency Target (100%)

3) Top 5/10 Events

4) Time Mode Statistics

The first section displayed on the report shows a summary of the snapshot window for your report as well as a brief look at the elapsed time, which represents the snapshot window, and the DB time, which represents activity on your database. If the **DB time exceeds the elapsed time**, it denotes a busy database. If it is a lot higher than the elapsed time, it may mean that some sessions are waiting for resources.

The instance efficiency section gives you a very quick view to determine if things are running adequately on your database. Generally, most percentages within this section should be above 90%. The Parse CPU to Parse Elapsd metric shows how much time the CPU is spending parsing SQL statements.

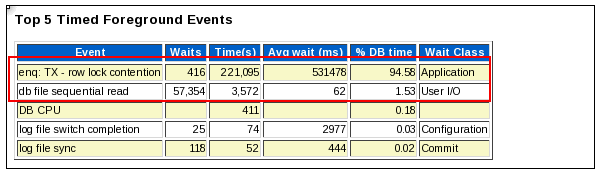
The third place to get a quick glance at your database performance is the Top 5 Timed Events

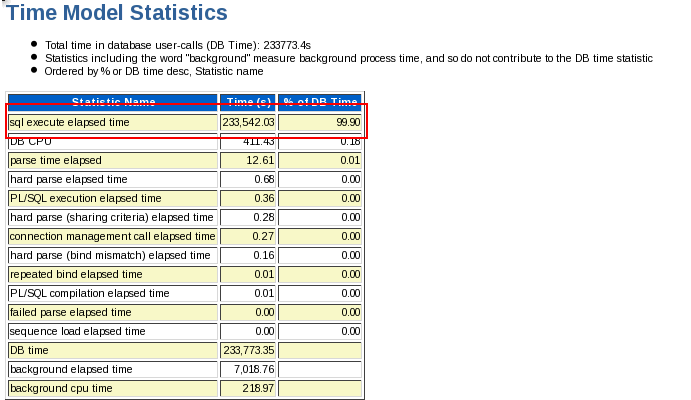
section. This section gives you a quick look at exactly where the highest amount of resources are being consumed within your database for the snapshot period. Based on these results, it may show you that

there is an inordinate amount of time spent performing full-table scans, or getting data across a network database link.

Below is one example which shows main point related to of awr report which are checked and recommend to application team.

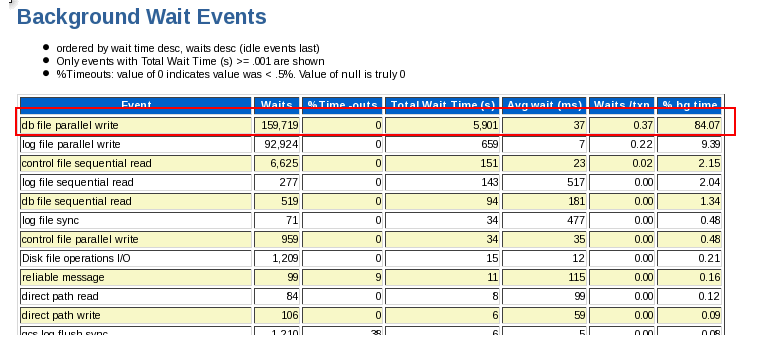
IN Top Wait Event is coming enq:TX - row lock contention its totally based on one Query which was being fired from App side.





It is also forcing us to go for the SQL Ordered part as sql execute to elapsed time is high close to 99% of DB time was being spent on it.

Let go on the Drill Down part now based on the above observation:-

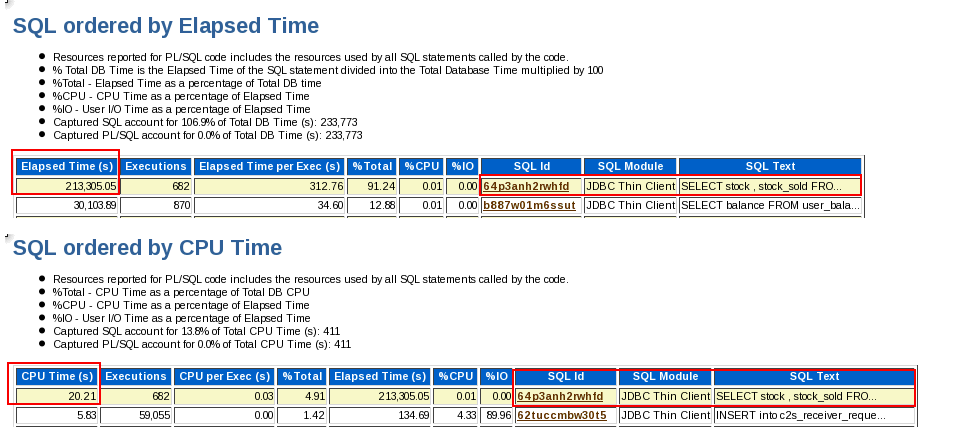


DB file parallel write wait event is totally related to I/O Problem , I will show the artifacts of the same below as well.

**Common Causes and Actions**

The db file parallel write latency is normally a symptom of a slow I/O subsystem or poor I/O configurations. This includes poor layout of database files, bad mount point to I/O controller ratio, wrong stripe size and/or RAID level, and not enough disks (i.e. there are a few high capacity disks versus many lower capacity disks).

The DBA needs to look at the average I/O time.



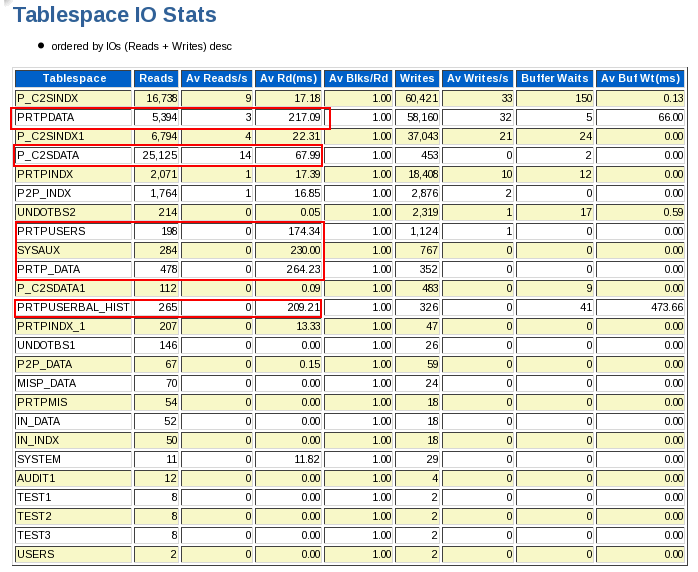
Please find the Query which is Creating a problem in the DB as its running with FOR Update clause also its coming high on the Elapsed Time as well as on the CPU Time.

So request you please check this Query and its logic with App Team.

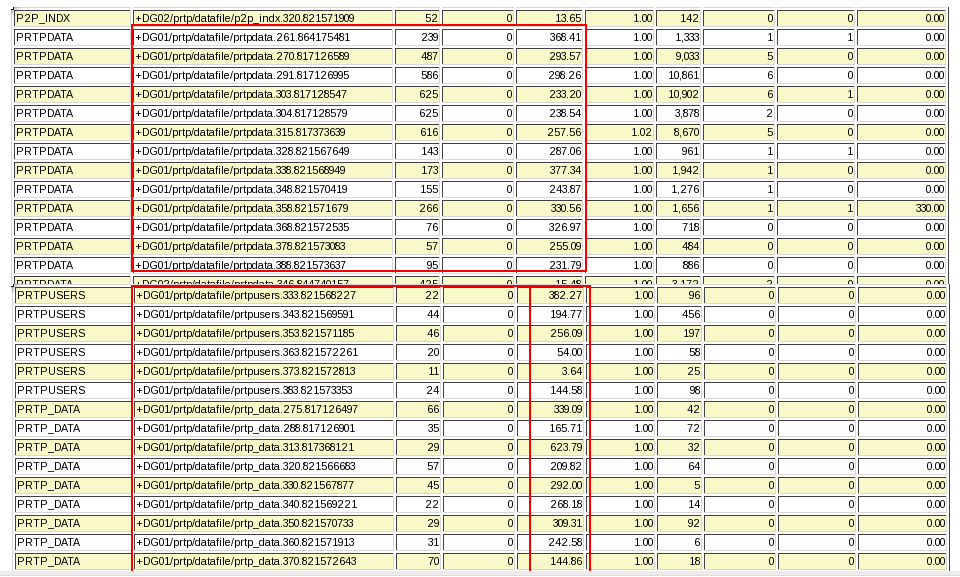


Here is the Artifacts of I/O Problem , Please check the Av RD(ms) column .

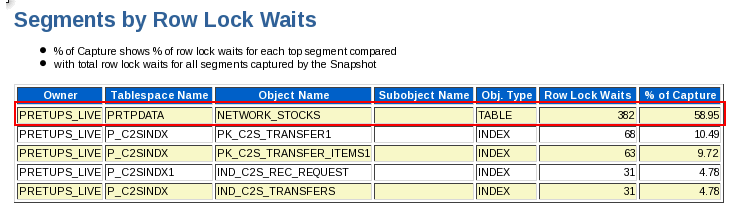
In a good I/O system it should not go above the 20 but in our case it is going much beyond that request you to take care this part asap..



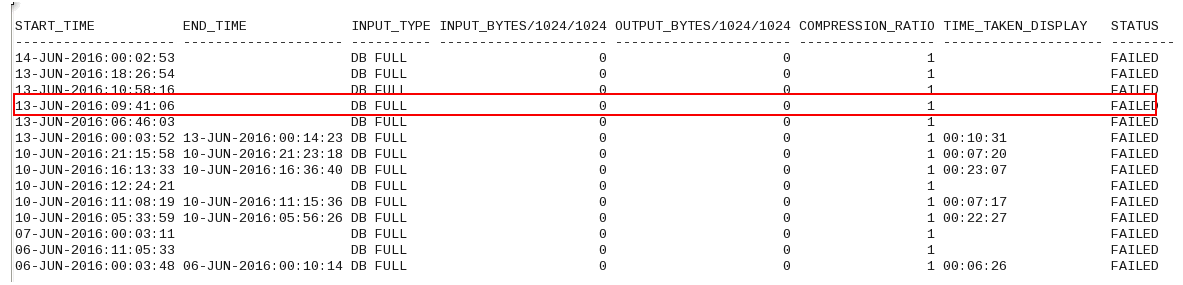
Files Snapshot :-



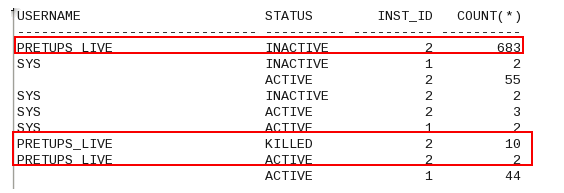
The Table used in the above Highlighted Query i.e Network\_Stocks table is the main object creating the Row Locks Waits as well.



Backup was also running during the Problematic time :-



Also when I checked the System I observed that All Application Session were going on the Node 2 only but today its looking good.



Real –Time AWR Analysis

Operating System Statistics

Statistic Total

AVG\_BUSY\_TIME 127,823

AVG\_IDLE\_TIME 360,597

AVG\_IOWAIT\_TIME 50,342

AVG\_SYS\_TIME 77,536 <---

AVG\_USER\_TIME 50,133

BUSY\_TIME 1,279,619

IDLE\_TIME 3,607,361

IOWAIT\_TIME 504,842

SYS\_TIME 776,815 <-- CPU usage in sys mode is higer than user mode cpu

USER\_TIME 502,804

Service Wait Class Stats

Service Name User I/O Total Wts User I/O Wt Time Concurcy Total Wts Concurcy Wt Time Admin Total Wts Admin Wt Time Network Total Wts Network Wt Time

APE 374814 2507 6258 10 0 0 7446511 10

SQL ordered by Reads

Physical Reads Executions Reads per Exec %Total CPU Time (s) Elapsed Time (s) SQL Id SQL Module SQL Text

154,993 1 154,993.00 14.84 13.46 78.82 0fzw3xs5kftff brconnect@hpx238 (TNS V1-V3) BEGIN DBMS\_STATS.GATHER\_TABLE\_...

Buffer Pool Advisory

Only rows with estimated physical reads >0 are displayed ordered by Block Size, Buffers For Estimate

P Size for Est (M) Size Factor Buffers for Estimate Est Phys Read Factor Estimated Physical Reads

D 208 0.09 25,727 1.56 2,863,218

D 416 0.19 51,454 1.31 2,398,979

D 624 0.28 77,181 1.22 2,241,790

D 832 0.37 102,908 1.16 2,125,522

D 1,040 0.47 128,635 1.11 2,040,142

D 1,248 0.56 154,362 1.08 1,985,035

D 1,456 0.65 180,089 1.06 1,940,507

D 1,664 0.75 205,816 1.04 1,899,354

D 1,872 0.84 231,543 1.02 1,866,894

D 2,080 0.94 257,270 1.01 1,844,630

D 2,224 1.00 275,081 1.00 1,831,660 <-- current value

D 2,288 1.03 282,997 1.00 1,824,321

D 2,496 1.12 308,724 0.99 1,809,369

D 2,704 1.22 334,451 0.98 1,791,970

D 2,912 1.31 360,178 0.96 1,758,676

D 3,120 1.40 385,905 0.94 1,726,093

D 3,328 1.50 411,632 0.93 1,710,225

D 3,536 1.59 437,359 0.93 1,699,210

D 3,744 1.68 463,086 0.92 1,676,303 <--- Can be seen benefit ,if increased

D 3,952 1.78 488,813 0.91 1,669,838

D 4,160 1.87 514,540 0.91 1,663,223

Observations

================

++ log switches are occuring 12 per hour. Please resize redo logs and add redo log more groups ,in order to ensure that log switch to happen at around 30 min interval.

++ Seeing large "Buffer Busy Waits" on object DBTABLOG~0,can be rebuild with large initrans(50) and pctfree (50)

++ Seeing zero free memory with buffer cache and shared pool. Please ensure to resize this pools

to get overall performance improvements.

++ "Operating System Statistics"says that sys space cpu usage is higher compared to user space cpu usage.

Please have OS admin to check any issue with OS functionality.In ideal practice user space cpu usage should be greater than or equal to sys space cpu usage

++ From "Service Wait Class Stats" ,the APE Service is showing large network waits.

Please check if there is any network latencies .

++ Seeing dbms\_stats gathering activity was in use by brconnect@hpx238 module.Please gather statistics

gathering activity during non-peak hours.

ASH Report

ASH Report(Oracle Active Session History)- it need to generate when performance analysis on sessions that run too frequently or are too short to be available on AWR  
snapshots. it can show more real-time or near real-time session information to assist in doing performance analysis on your database. The ASH default sample is 1 second, but later is stored in 10 second intervals on disk.

it is useful to collect ASH reports in a situations where you need to narrow down which selects are responsible for a particular wait

you want to know when a particular wait occurred within the snapshot period to tie up with performance spikes or intermittent hang

Using ASH, Yoy will get following

* Top SQL
* Top Sessions
* Top Waits
* Blocking Sessions
* Top Objects
* Waits by time during sample intervals

SQL>script for getting ASH Report on RAC database:

SQL>@$ORACLE\_HOME/rdbms/admin/ashrpti.sql

SQL script for getting ASH Report for single Instance:

SQL>@$ORACLE\_HOME/rdbms/admin/ashrpt.sql

**ADDM Report**

The Automatic Database Diagnostic Monitor (ADDM) analyzes data in the Automatic Workload Repository (AWR) to identify potential performance bottlenecks. For each of the identified issues it locates the root cause and provides recommendations for correcting the problem. An ADDM analysis task is performed and its findings and recommendations stored in the database every time an AWR snapshot is taken provided the STATISTICS\_LEVEL parameter is set to TYPICAL or ALL. The ADDM analysis includes the following.

* CPU load
* Memory usage
* I/O usage
* Resource intensive SQL
* Resource intensive PL/SQL and Java
* RAC issues
* Application issues
* Database configuration issues
* Concurrency issues
* Object contention

## addmrpt.sql Script

The addmrpt.sql script can be used to create an ADDM report from SQL\*Plus. The script is called as follows.

-- UNIX

@/u01/app/oracle/product/10.1.0/db\_1/rdbms/admin/addmrpt.sql

-- Windows

@d:\oracle\product\10.1.0\db\_1\rdbms\admin\addmrpt.sql

SQL> @?/rdbms/admin/addmrpt.sql

**Below is one test case for addm:-**

I found row lock wait was too high so down the line found problematic query as well and shared that query with application team to check from their and it was rectified by them.

addmrpt\_1\_22542\_22543.txt

ADDM Report for Task 'TASK\_33939'

---------------------------------

Analysis Period

---------------

AWR snapshot range from 22542 to 22543.

Time period starts at 06-JUN-16 04.00.06 PM

Time period ends at 06-JUN-16 05.00.12 PM

Analysis Target

---------------

Database 'DBNAME' with DB ID 577252600.

Database version 11.2.0.3.0.

ADDM performed an analysis of instance PACSMG, numbered 1 and hosted at

XXXXXXXX.

Activity During the Analysis Period

-----------------------------------

Total database time was 36014 seconds.

The average number of active sessions was 9.99.

Summary of Findings

-------------------

Description Active Sessions Recommendation

s

Percent of Activity

---------------------------------------- ------------------- --------------

-

1 Top SQL Statements 6.59 | 66.02 6

**2 Row Lock Waits 4.17 | 41.7 1**

3 Undersized SGA .64 | 6.4 1

4 Top Segments by "User I/O" and "Cluster" .19 | 1.91 1

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Findings and Recommendations

----------------------------

Finding 1: Top SQL Statements

Impact is 6.59 active sessions, 66.02% of total activity.

---------------------------------------------------------

SQL statements consuming significant database time were found. These

statements offer a good opportunity for performance improvement.

Recommendation 1: SQL Tuning

Estimated benefit is 4.46 active sessions, 44.68% of total activity.

--------------------------------------------------------------------

Action

Investigate the SELECT statement with SQL\_ID "8wfzx3nyvma8m" for

possible performance improvements. You can supplement the information

given here with an ASH report for this SQL\_ID.

Related Object

SQL statement with SQL\_ID 8wfzx3nyvma8m.

SELECT TRIM(MDL\_PARAMETER\_VALUE) FROM FDS\_M\_MODULE\_PARAM WHERE

MOD\_1\_MOD\_ID ='PROVS' AND MDL\_PARAMETER\_ID =:B1 FOR UPDATE OF

FDS\_M\_MODULE\_PARAM.MDL\_PARAMETER\_VALUE

Rationale

The SQL spent only 0% of its database time on CPU, I/O and Cluster

waits. Therefore, the SQL Tuning Advisor is not applicable in this case.

Look at performance data for the SQL to find potential improvements.

Rationale

Database time for this SQL was divided as follows: 100% for SQL

execution, 0% for parsing, 0% for PL/SQL execution and 0% for Java

execution.

Rationale

SQL statement with SQL\_ID "8wfzx3nyvma8m" was executed 412 times and had

an average elapsed time of 36 seconds.

Rationale

Waiting for event "enq: TX - row lock contention" in wait class

"Application" accounted for 100% of the database time spent in

processing the SQL statement with SQL\_ID "8wfzx3nyvma8m".

Recommendation 2: SQL Tuning

Estimated benefit is .54 active sessions, 5.41% of total activity.

------------------------------------------------------------------

Action

Run SQL Tuning Advisor on the UPSERT statement with SQL\_ID

"672j4rmvvnmru".

Related Object

SQL statement with SQL\_ID 672j4rmvvnmru.

MERGE INTO SIMSWAP\_REPROCESS X USING ( SELECT SYSDATE CREATION\_DATE,

B.SER\_SERVICE\_ORDER\_ID, A.SER\_MSISDN, A.SER\_IMSI\_NO,

A.SER\_SERVICE\_VALUE, B.SER\_RECIEVED\_DT\_TIME, A.SER\_ERROR, 'PENDING'

STATUS, NULL HLR\_IMSI, NULL REPROCESS\_DATE, D.SIM\_SIM\_NO NEW\_SIM\_NO

FROM INCMS.CIN\_M\_SIM D, INCMS.CIN\_T\_SERVICE\_ORDER\_DETAIL A,

INCMS.CIN\_T\_SERVICE\_ORDER B, ( SELECT SER\_MSISDN,

MAX(SER\_RECIEVED\_DT\_TIME) SER\_RECIEVED\_DT\_TIME FROM

INCMS.CIN\_T\_SERVICE\_ORDER\_DETAIL WHERE SER\_RECIEVED\_DT\_TIME >

SYSDATE-3 AND SER\_SERVICE\_ID IN ('SIMCH','SMCHE') GROUP BY SER\_MSISDN

) C WHERE B.SER\_SERVICE\_ORDER\_ID = A.SER\_SERVICE\_ORDER\_ID AND

D.SIM\_IMSI\_NO = SUBSTR(A.SER\_SERVICE\_VALUE,1,15) AND B.SER\_OPERATION

= 'MODS' AND B.SER\_PROVISIONING\_TYPE <>'BULK' AND

A.SER\_RECIEVED\_DT\_TIME > SYSDATE-3 AND A.SER\_SERVICE\_ID IN

('SIMCH','SMCHE') AND A.SER\_MSISDN = C.SER\_MSISDN AND

A.SER\_RECIEVED\_DT\_TIME = C.SER\_RECIEVED\_DT\_TIME AND A.SER\_STATUS =

'FAIL' AND (A.SER\_ERROR LIKE 'Unknown Error:RESP:18310%' OR

A.SER\_ERROR = 'Ext. system communication link failure.Please contact

switch people.' OR A.SER\_ERROR = 'Unknown Error:RESP:18304 recieved

from EMA.Please Contact switch people' OR A.SER\_ERROR LIKE '%fail

Fail File of Zero Size' OR A.SER\_ERROR LIKE 'Unknown

Error:RESP:18398%' ) ) Y ON (X.SER\_SERVICE\_ORDER\_ID =

Y.SER\_SERVICE\_ORDER\_ID) WHEN NOT MATCHED THEN INSERT(X.CREATION\_DATE,

X.SER\_SERVICE\_ORDER\_ID, X.SER\_MSISDN, X.SER\_IMSI\_NO,

X.SER\_SERVICE\_VALUE, X.SER\_RECIEVED\_DT\_TIME, X.SER\_ERROR, X.STATUS)

VALUES(SYSDATE, Y.SER\_SERVICE\_ORDER\_ID, Y.SER\_MSISDN, Y.SER\_IMSI\_NO,

Y.SER\_SERVICE\_VALUE, Y.SER\_RECIEVED\_DT\_TIME, Y.SER\_ERROR, 'PENDING')

WHEN MATCHED THEN UPDATE SET X.CREATION\_DATE = SYSDATE, X.STATUS =

'PENDING'

Rationale

The SQL spent 100% of its database time on CPU, I/O and Cluster waits.

This part of database time may be improved by the SQL Tuning Advisor.

Rationale

Database time for this SQL was divided as follows: 100% for SQL

execution, 0% for parsing, 0% for PL/SQL execution and 0% for Java

execution.

Rationale

SQL statement with SQL\_ID "672j4rmvvnmru" was executed 109 times and had

an average elapsed time of 17 seconds.

Rationale

Top level calls to execute the PL/SQL statement with SQL\_ID

"g2j4rw7vpbcbm" are responsible for 100% of the database time spent on

the UPSERT statement with SQL\_ID "672j4rmvvnmru".

Related Object

SQL statement with SQL\_ID g2j4rw7vpbcbm.

BEGIN provisioning.simswap\_reprocess; END;

Recommendation 3: SQL Tuning

Estimated benefit is .53 active sessions, 5.26% of total activity.

------------------------------------------------------------------

Action

Run SQL Tuning Advisor on the SELECT statement with SQL\_ID

"52us6knb8uw7a".

Related Object

SQL statement with SQL\_ID 52us6knb8uw7a.

SELECT TO\_CHAR("A1"."START\_TIME",:"SYS\_B\_00"),TO\_CHAR("A1"."START\_TIM

E",:"SYS\_B\_01"),TO\_CHAR("A1"."START\_TIME",:"SYS\_B\_02"),"A1"."APN\_NI",

CASE "A1"."SGSN\_ADDRESS" WHEN :"SYS\_B\_03" THEN :"SYS\_B\_04" ELSE CASE

WHEN ("A1"."ORIGINAL\_CALL\_TYPE"=:"SYS\_B\_05" AND "A1"."SGSN\_ADDRESS"

IS NULL) THEN :"SYS\_B\_06" ELSE :"SYS\_B\_07" END END

,COUNT(\*),SUM(NVL("A1"."DATA\_UP",:"SYS\_B\_08")),SUM(NVL("A1"."DATA\_DOW

N",:"SYS\_B\_09")),SUM(NVL("A1"."DATA\_UP",:"SYS\_B\_10"))+SUM(NVL("A1"."D

ATA\_DOWN",:"SYS\_B\_11")),SUM(NVL("A1"."DATA\_UP",:"SYS\_B\_12"))/:"SYS\_B\_

13"+SUM(NVL("A1"."DATA\_DOWN",:"SYS\_B\_14"))/:"SYS\_B\_15",:"SYS\_B\_16",CA

SE "A1"."SGSN\_ADDRESS" WHEN :"SYS\_B\_17" THEN :"SYS\_B\_18" ELSE CASE

WHEN ("A1"."ORIGINAL\_CALL\_TYPE"=:"SYS\_B\_19" AND "A1"."SGSN\_ADDRESS"

IS NULL) THEN :"SYS\_B\_20" ELSE :"SYS\_B\_21" END END FROM

"INCMS"."HBST\_RAT\_UNBILD\_CALL""A1" WHERE

"A1"."SERVICE\_ID"=:"SYS\_B\_22" AND

TO\_NUMBER(TO\_CHAR("A1"."START\_TIME",:"SYS\_B\_23"))=:"SYS\_B\_24" GROUP

BY TO\_CHAR("A1"."START\_TIME",:"SYS\_B\_25"),TO\_CHAR("A1"."START\_TIME",:

"SYS\_B\_26"),TO\_CHAR("A1"."START\_TIME",:"SYS\_B\_27"),"A1"."APN\_NI","A1"

."SGSN\_ADDRESS","A1"."ORIGINAL\_CALL\_TYPE"

Rationale

The SQL spent 100% of its database time on CPU, I/O and Cluster waits.

This part of database time may be improved by the SQL Tuning Advisor.

Rationale

Database time for this SQL was divided as follows: 100% for SQL

execution, 0% for parsing, 0% for PL/SQL execution and 0% for Java

execution.

Rationale

SQL statement with SQL\_ID "52us6knb8uw7a" was executed 4 times and had

an average elapsed time of 446 seconds.

Recommendation 4: SQL Tuning

Estimated benefit is .41 active sessions, 4.1% of total activity.

-----------------------------------------------------------------

Action

Run SQL Tuning Advisor on the SELECT statement with SQL\_ID

"cxwc6bnrmapfh".

Related Object

SQL statement with SQL\_ID cxwc6bnrmapfh.

select b.ser\_service\_order\_id||:"SYS\_B\_0"||a.ser\_msisdn||:"SYS\_B\_1"||

a.ser\_service\_value||:"SYS\_B\_2"||a.ser\_error

from incms.CIN\_T\_SERVICE\_ORDER\_DETAIL a, incms.CIN\_T\_SERVICE\_ORDER b,

incms.cms\_m\_customer c

where b.SER\_SERVICE\_ORDER\_ID = a.SER\_SERVICE\_ORDER\_ID

and a.ser\_msisdn = c.cus\_tel\_no

and b.ser\_provisioning\_type = :"SYS\_B\_3"

and b.ser\_operation in (:"SYS\_B\_4")

and a.SER\_RECIEVED\_DT\_TIME > sysdate-:"SYS\_B\_5"

and a.SER\_SERVICE\_ID in (:"SYS\_B\_6",:"SYS\_B\_7")

and a.ser\_status = :"SYS\_B\_8"

order by b.ser\_service\_order\_id

Rationale

The SQL spent 100% of its database time on CPU, I/O and Cluster waits.

This part of database time may be improved by the SQL Tuning Advisor.

Rationale

Database time for this SQL was divided as follows: 100% for SQL

execution, 0% for parsing, 0% for PL/SQL execution and 0% for Java

execution.

Rationale

SQL statement with SQL\_ID "cxwc6bnrmapfh" was executed 81 times and had

an average elapsed time of 17 seconds.

Recommendation 5: SQL Tuning

Estimated benefit is .41 active sessions, 4.07% of total activity.

------------------------------------------------------------------

Action

Run SQL Tuning Advisor on the SELECT statement with SQL\_ID

"an6y0xhff8hsc".

Related Object

SQL statement with SQL\_ID an6y0xhff8hsc.

select account\_no,invoice\_date,os,invoice\_Date+:"SYS\_B\_0",floor(sysda

te-(invoice\_date+:"SYS\_B\_1")),unadjusted\_amount,add\_months(invoice\_da

te,-:"SYS\_B\_2") from hbst\_dunning\_os

Rationale

The SQL spent 100% of its database time on CPU, I/O and Cluster waits.

This part of database time may be improved by the SQL Tuning Advisor.

Rationale

Database time for this SQL was divided as follows: 100% for SQL

execution, 0% for parsing, 0% for PL/SQL execution and 0% for Java

execution.

Rationale

SQL statement with SQL\_ID "an6y0xhff8hsc" was executed 54 times and had

an average elapsed time of 25 seconds.

Rationale

At least 2 distinct execution plans were utilized for this SQL statement

during the analysis period.

Recommendation 6: SQL Tuning

Estimated benefit is .25 active sessions, 2.5% of total activity.

-----------------------------------------------------------------

Action

Run SQL Tuning Advisor on the SELECT statement with SQL\_ID

"5rs41gbgw5x10".

Related Object

SQL statement with SQL\_ID 5rs41gbgw5x10.

SELECT ceq\_msg\_id, ceq\_email\_id, ceq\_subject, ceq\_msg\_body,

CSQ\_ATTACHMENT FROM cin\_t\_email\_queue WHERE ceq\_status = :"SYS\_B\_0"

and rownum< (select mdl\_parameter\_value from fds\_m\_module\_param where

mod\_1\_mod\_id=:"SYS\_B\_1" and mdl\_parameter\_id=:"SYS\_B\_2")

Rationale

The SQL spent 100% of its database time on CPU, I/O and Cluster waits.

This part of database time may be improved by the SQL Tuning Advisor.

Rationale

Database time for this SQL was divided as follows: 100% for SQL

execution, 0% for parsing, 0% for PL/SQL execution and 0% for Java

execution.

Rationale

SQL statement with SQL\_ID "5rs41gbgw5x10" was executed 266 times and had

an average elapsed time of 3 seconds.

Rationale

Full scan of TABLE "INCMS.CIN\_T\_EMAIL\_QUEUE" with object ID 83944

consumed 97% of the database time spent on this SQL statement.

Finding 2: Row Lock Waits

Impact is 4.17 active sessions, 41.7% of total activity.

--------------------------------------------------------

SQL statements were found waiting for row lock waits.

Recommendation 1: Application Analysis

Estimated benefit is 4.16 active sessions, 41.68% of total activity.

--------------------------------------------------------------------

Action

Significant row contention was detected in the TABLE

"INCMS.FDS\_M\_MODULE\_PARAM" with object ID 84070. Trace the cause of row

contention in the application logic using the given blocked SQL.

Related Object

Database object with ID 84070.

Rationale

The SQL statement with SQL\_ID "8wfzx3nyvma8m" was blocked on row locks.

Related Object

SQL statement with SQL\_ID 8wfzx3nyvma8m.

SELECT TRIM(MDL\_PARAMETER\_VALUE) FROM FDS\_M\_MODULE\_PARAM WHERE

MOD\_1\_MOD\_ID ='PROVS' AND MDL\_PARAMETER\_ID =:B1 FOR UPDATE OF

FDS\_M\_MODULE\_PARAM.MDL\_PARAMETER\_VALUE

Rationale

The session with ID 881 and serial number 40733 in instance number 1 was

the blocking session responsible for 57% of this recommendation's

benefit.

Rationale

The session with ID 94 and serial number 57631 in instance number 1 was

the blocking session responsible for 42% of this recommendation's

benefit.

Symptoms That Led to the Finding:

---------------------------------

Wait class "Application" was consuming significant database time.

Impact is 4.25 active sessions, 42.58% of total activity.

Finding 3: Undersized SGA

Impact is .64 active sessions, 6.4% of total activity.

------------------------------------------------------

The SGA was inadequately sized, causing additional I/O or hard parses.

The value of parameter "sga\_target" was "12288 M" during the analysis period.

Recommendation 1: Database Configuration

Estimated benefit is .32 active sessions, 3.22% of total activity.

------------------------------------------------------------------

Action

Increase the size of the SGA by setting the parameter "sga\_target" to

13824 M.

Symptoms That Led to the Finding:

---------------------------------

Wait class "User I/O" was consuming significant database time.

Impact is 1.26 active sessions, 12.6% of total activity.

Finding 4: Top Segments by "User I/O" and "Cluster"

Impact is .19 active sessions, 1.91% of total activity.

-------------------------------------------------------

Individual database segments responsible for significant "User I/O" and

"Cluster" waits were found.

Recommendation 1: Segment Tuning

Estimated benefit is .19 active sessions, 1.91% of total activity.

------------------------------------------------------------------

Action

Run "Segment Advisor" on TABLE "INCMS.CIN\_T\_EMAIL\_QUEUE" with object ID

83944.

Related Object

Database object with ID 83944.

Action

Investigate application logic involving I/O on TABLE

"INCMS.CIN\_T\_EMAIL\_QUEUE" with object ID 83944.

Related Object

Database object with ID 83944.

Action

Look at the "Top SQL Statements" finding for SQL statements consuming

significant I/O on this segment. For example, the SELECT statement with

SQL\_ID "5rs41gbgw5x10" is responsible for 100% of "User I/O" and

"Cluster" waits for this segment.

Rationale

The I/O usage statistics for the object are: 0 full object scans,

18116994 physical reads, 0 physical writes and 18116994 direct reads.

Symptoms That Led to the Finding:

---------------------------------

Wait class "User I/O" was consuming significant database time.

Impact is 1.26 active sessions, 12.6% of total activity.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Additional Information

----------------------

Miscellaneous Information

-------------------------

Wait class "Commit" was not consuming significant database time.

Wait class "Concurrency" was not consuming significant database time.

Wait class "Configuration" was not consuming significant database time.

CPU was not a bottleneck for the instance.

Wait class "Network" was not consuming significant database time.

Session connect and disconnect calls were not consuming significant database

time.

Hard parsing of SQL statements was not consuming significant database time.

### Use of ADDM Reports alongside AWR

ADDM reports can be reviewed along with AWR to assist in diagnosis since they provide specific recommendations which can help point at potential problems. The following is a sample ADDM report taken from:

[Document 250655.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1359094.1&id=250655.1)How to use the Automatic Database Diagnostic Monitor:

Example Output:

DETAILED ADDM REPORT FOR TASK 'SCOTT\_ADDM' WITH ID 5  
----------------------------------------------------  
  
Analysis Period: 17-NOV-2003 from 09:50:21 to 10:35:47  
Database ID/Instance: 494687018/1  
Snapshot Range: from 1 to 3  
Database Time: 4215 seconds  
Average Database Load: 1.5 active sessions  
  
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~  
  
  
FINDING 1: 65% impact (2734 seconds)  
------------------------------------  
PL/SQL execution consumed significant database time.  
  
RECOMMENDATION 1: SQL Tuning, 65% benefit (2734 seconds)  
ACTION: Tune the PL/SQL block with SQL\_ID fjxa1vp3yhtmr. Refer to   
the "Tuning PL/SQL Applications" chapter of Oracle's "PL/SQL   
User's Guide and Reference"  
RELEVANT OBJECT: SQL statement with SQL\_ID fjxa1vp3yhtmr  
BEGIN EMD\_NOTIFICATION.QUEUE\_READY(:1, :2, :3); END;  
  
FINDING 2: 35% impact (1456 seconds)  
------------------------------------  
SQL statements consuming significant database time were found.  
  
RECOMMENDATION 1: SQL Tuning, 35% benefit (1456 seconds)  
ACTION: Run SQL Tuning Advisor on the SQL statement with SQL\_ID  
gt9ahqgd5fmm2.  
RELEVANT OBJECT: SQL statement with SQL\_ID gt9ahqgd5fmm2 and  
PLAN\_HASH 547793521  
UPDATE bigemp SET empno = ROWNUM  
  
FINDING 3: 20% impact (836 seconds)  
-----------------------------------  
The throughput of the I/O subsystem was significantly lower than expected.  
  
RECOMMENDATION 1: Host Configuration, 20% benefit (836 seconds)  
ACTION: Consider increasing the throughput of the I/O subsystem.  
Oracle's recommended solution is to stripe all data file using   
the SAME methodology. You might also need to increase the   
number of disks for better performance.  
  
RECOMMENDATION 2: Host Configuration, 14% benefit (584 seconds)  
ACTION: The performance of file   
D:\ORACLE\ORADATA\V1010\UNDOTBS01.DBF was significantly worse   
than other files. If striping all files using the SAME   
methodology is not possible, consider striping this file over   
multiple disks.  
RELEVANT OBJECT: database file  
"D:\ORACLE\ORADATA\V1010\UNDOTBS01.DBF"  
  
SYMPTOMS THAT LED TO THE FINDING:  
Wait class "User I/O" was consuming significant database time.   
(34% impact [1450 seconds])  
  
FINDING 4: 11% impact (447 seconds)  
-----------------------------------  
Undo I/O was a significant portion (33%) of the total database I/O.  
  
NO RECOMMENDATIONS AVAILABLE  
  
SYMPTOMS THAT LED TO THE FINDING:  
The throughput of the I/O subsystem was significantly lower than  
expected. (20% impact [836 seconds])  
Wait class "User I/O" was consuming significant database time.   
(34% impact [1450 seconds])  
  
FINDING 5: 9.9% impact (416 seconds)  
------------------------------------  
Buffer cache writes due to small log files were consuming significant   
database time.  
  
RECOMMENDATION 1: DB Configuration, 9.9% benefit (416 seconds)  
ACTION: Increase the size of the log files to 796 M to hold at   
least 20 minutes of redo information.

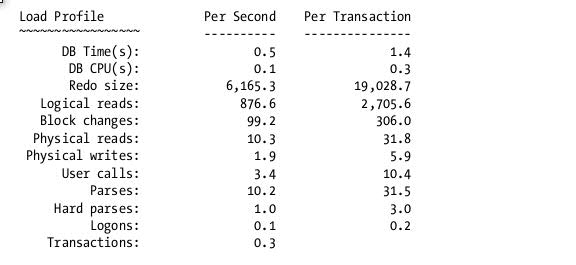
ADDM report gives possible recommendations in more readable format than AWR.  However, ADDM should be interpreted along with AWR statistics for accurate diagnostics.

**Oracle Performance Tuning Technique**

**Improving performance means**  
  
A) Reduce response time  
B) Increase throughput   
  
**How to start Tuning?**   
  
A) How frequently the same issue occurred as of now?   
B) is every time, the issue reported time is same?   
C) is the issue only from specific user?   
D) is the issue only from specific location?   
E) is the issue only from specific Machine?   
F) is the issue only from specific Application?   
G) are there any changes happened in the Applications/Databases/Query   
 O/s, H/w, Network   
H) is the issue is in Business Hours/ non-business hours?   
I) are there any backups jobs running at the time of problem?   
J) whether latest statistics collected or not?   
K) are there any increase in number of users/calls/transactions?  
  
**Tuning Tool**  
  
1) Automated Maintenance Tasks   
2) ADDM Report / Regular ADDM Report (pre-12c)   
3) ADDM Compare Report (New in 12c)  
4) Real-Time ADDM Report (New in 12c)   
5) Emergency Monitoring (Enhacement in "Memory Access Mode")   
6) ASH Report and AWR Report   
  
  
**Automated Maintenance Tasks**  
  
Oracle 11g includes three automated database maintenance tasks:  
  
**Automatic Optimizer Statistics Collection** - Gathers stale or missing statistics for all schema objects . The task name is 'auto optimizer stats collection'.  
 **Automatic Segment Advisor** - Identifies segments that could be reorganized to save space. The task name is 'auto space advisor'.  
**Automatic SQL Tuning Advisor** - Identifies and attempts to tune high load SQL. The task name is 'sql tuning advisor'.  
  
These tasks run during maintenance windows scheduled to open over night.   
  
 set lines 180 pages 1000  
set lines 180 pages 1000  
col client\_name for a40  
col attributes for a60  
select client\_name, status,attributes,service\_name from dba\_autotask\_client  
/  
SQL >  
CLIENT\_NAME STATUS ATTRIBUTES SERVICE\_NAME  
---------------------------------------- -------- ------------------------------------------------------------   
auto optimizer stats collection DISABLED ON BY DEFAULT, VOLATILE, SAFE TO KILL  
auto space advisor ENABLED ON BY DEFAULT, VOLATILE, SAFE TO KILL  
sql tuning advisor ENABLED ONCE PER WINDOW, ON BY DEFAULT, VOLATILE, SAFE TO KILL  
  
  
SELECT count(\*) FROM dba\_autotask\_client\_history WHERE client\_name = 'auto optimizer stats collection' AND window\_start\_time > (SYSDATE-8);  
  
SELECT CLIENT\_NAME,JOBS\_STARTED,JOBS\_COMPLETED,WINDOW\_END\_TIME FROM dba\_autotask\_client\_history WHERE client\_name = 'auto optimizer stats collection' AND window\_start\_time > (SYSDATE-8);  
  
CLIENT\_NAME JOBS\_STARTED JOBS\_COMPLETED WINDOW\_END\_TIME  
---------------------------------------------------------------- ------------ -------------- ---------------------------------------------------------------------------  
auto optimizer stats collection                                             1              1 29-OCT-16 03.00.00.015508 AM +01:00  
auto optimizer stats collection                                             1              1 01-NOV-16 02.00.00.020355 AM +00:00  
auto optimizer stats collection                                             5              5 30-OCT-16 02.00.00.009613 AM +00:00  
auto optimizer stats collection                                             5              5 31-OCT-16 02.00.00.006245 AM +00:00  
auto optimizer stats collection                                             1              1 28-OCT-16 03.00.00.011625 AM +01:00  
auto optimizer stats collection                                             1              1 26-OCT-16 03.00.00.026329 AM +01:00  
auto optimizer stats collection                                             1              1 02-NOV-16 02.00.00.005620 AM +00:00  
auto optimizer stats collection                                             1              1 27-OCT-16 03.00.00.031229 AM +01:00  
 **To collect complete Info**  
  
ALTER SESSION SET NLS\_TIMESTAMP\_TZ\_FORMAT ='DD/MM/YYYY HH24:MI:SS TZR TZD';  
ALTER SESSION SET NLS\_DATE\_FORMAT ='DD-MM-YYYY HH24:MI:SS';  
set pagesize 9999  
spool c:\dba\_autotask\_client.html  
set markup html on  
select \* from DBA\_AUTOTASK\_CLIENT;  
select \* from DBA\_AUTOTASK\_CLIENT\_HISTORY;  
select \* from DBA\_AUTOTASK\_CLIENT\_JOB;  
select \* from DBA\_AUTOTASK\_JOB\_HISTORY order by JOB\_START\_TIME;  
select \* from DBA\_AUTOTASK\_OPERATION;  
select \* from DBA\_AUTOTASK\_SCHEDULE order by START\_TIME;  
select \* from DBA\_AUTOTASK\_TASK;  
select \* from DBA\_AUTOTASK\_WINDOW\_CLIENTS;  
select \* from DBA\_AUTOTASK\_WINDOW\_HISTORY order by WINDOW\_START\_TIME;  
select \* from dba\_scheduler\_windows;  
select \* from dba\_scheduler\_window\_groups;  
select \* from dba\_scheduler\_job\_run\_details order by ACTUAL\_START\_DATE;  
select \* from DBA\_SCHEDULER\_JOB\_LOG;  
SELECT program\_name, program\_action, enabled FROM dba\_scheduler\_programs;  
spool off  
  
  
**ADDM Report / Regular ADDM Report (pre-12c)**  
  
 - auto generated report (system generated)   
 - user generated (Manually created ADDM)   
 - depended on Snapshots (AWR Based Report)   
 - findings are two categories   
 - for SQL Queries   
 - for waits at Database (Ex:- Memory structures)   
- addmrpt.sql   
- addmrpti.sql

- dbms\_advisor.get\_task\_report() from SQL Prompt   
- ADDM information can be contorled by statistics\_level   
- statistics\_level=basic => NO auto ADDM Report   
- ADDM Report for I/O Analysis partially depended on DBIO\_EXPECTED param   
  
NOTE:- DBIO\_EXPECTED = average time to read a single Database Block in micro sec.   
 5000 to 20000   
 default is 10000   
  
  
**ADDM Compare Report (New in 12c)**   
  
 - based on AWR Snapshots   
 - identifies the major changes for performance impact   
 - runs analysis for the base period and compare period   
 - snap1, snap2 => good time   
 - snap3, snap4 => bad time   
 - dbms\_addm is new PL/SQL Package   
 - compare report generation is possible   
  
  
**Real-Time ADDM Report (New in 12c)**  
  
 - based on ASH Data   
- ASH Data for last 10 minutes (available in SGA)   
 - did NOT find any blokings   
 - able to connect as sysdba   
 - "Emergency Monitoring" report NOT in a position to give root cause   
- PROD Performance is highly sick/slow   
 - generating ADDM is emergency   
  
  
**Emergency Monitoring (Enhancement in "Memory Access Mode")**   
  
 - useful at the time of PROD highly sick and not able to login   
- available in 11g R2 with the name "Memory Access Mode"  
- can be enabled/disabled   
- if enabled, collectors are working   
 - these collector collects information directly from SGA   
 - from 12c, it is "Emergency Monitoring"  
 - collecting information from SGA directly by "Agent"  
 - this information sent to OEM Single Page   
- that page has "hang analysis information", blockings,   
 kill option, shutdown options, etc.   
  
  
**ASH Report and AWR Report**   
  
ASH Report(Oracle Active Session History)- it need to generate when performance analysis on sessions that run too frequently or are too short to be available on AWR  
snapshots. The AWR snapshots are not taken often enough to capture the information that you need. it is sampled every second from V$SESSION, and it can show more real-time or near real-time session information to assist in doing performance analysis on your database.  
  
  
**Important Summary Sections in AWR Report (or) How to Read AWR Report?**

If DBTime is greater during bad period, it is likely the database is causing the problem; you have verified the problem is in the database

1) Load Profile => load on Instance / Instance activities   
2) Instance Efficiency Target (100%)   
3) Top 5/10 Events -=> high performance impact wait events   
4) Time Mode Statistics => to find root cause of Query issues  
5) Instance Efficiency Target (100%) => overall health of Database   
  
Based upon the relative information in above sections,   
 Go to Drill Down Sections  
  
Load Profile:-   
-------------   
Parses   
Hard Parses  
Physical Reads   
Physical Writes  
  
Instance Efficiency Target (100%):-   
----------------------------------

Buffer Hit%  
Library Hit%   
Soft Parse%   
Execute to Parse%   
Parse CPU to Parse Elapsed%

Where:

. Redo size: This is the amount of redo generated during this report.

. Logical Reads: This is calculated as Consistent Gets + DB Block Gets = Logical Reads

. Block changes: The number of blocks modified during the sample interval

. Physical Reads: The number of requests for a block that caused a physical I/O.

. Physical Writes: The number of physical writes issued.

. User Calls: The number of queries generated

. Parses: Total of all parses: both hard and soft

. Hard Parses: Those parses requiring a completely new parse of the SQL statement. These consume both latches and shared pool area.

. Soft Parses: Not listed but derived by subtracting the hard parses from parses. A soft parse reuses a previous hard parse and hence consumes far fewer resources.

. Sorts, Logons, Executes and Transactions are all self explanatory

Top 5/10 Events:-    
----------------  
Buffer Busy Waits   
Free Buffer Waits   
Latch: Cache Buffer Chains   
DB File Scatter Read   
DB File Sequential Read   
DB File Parallel Read   
  
(in case of Shared Pool or Library Cache Size issues)   
Latch: shared pool   
Latch: row cache objects   
  
Time Model Statistics:-   
----------------------   
Parse Time Elapsed   
Hard Parse Elapsed Time   
  
NOTE:- shared\_pool\_reserved\_size > 50% of shared\_pool\_size => signal memory leak error   
=> ORA-04031  
  
  
  
Free Buffer Waits:-   
  
Problem1:- buffer cache is too small   
  
Possible solutions:-   
  
1) use AMM (memory\_target) or ASMM (sga\_target)   
2) increase buffer cache size as per   
 a) ADDM Recommendations   
 b) AWR Buffer Cache Advisory Results   
c) V$db\_cache\_advice   
  
Problem2:- DBWR is slow   
  
Possible Solutions:-   
  
1) increase db\_writer\_processes => increases I/O bandwidth   
 (check with System Administrator for better value)   
  
2) increase I/O bandwidth by stripping the datafiles across multiple Disks (storage level)   
  
Buffer Busy Waits:-   
  
Find hot blocks / hot segments from   
  
segment statistics   
Segments by Buffer Busy Waits   
  
Solution:-   
  
1) Manually keep hot object into KEEP Buffer   
2) check the segment advisory results   
3) check the ADDM Recommenations   
4) check the Tablespace management (whether it is dictionary manager or locally managed)   
  
  
Latch: cache buffer chains:-   
---------------------------   
  
Problem1:- the cache chain latches are used when searching for adding or deleting   
 a buffer from the buffer cache chain   
  
Reason:- multiple users are trying to read code / description from look up tables   
  
Solutions:-   
  
a) identify the hot block   
b) modify the application to use PL/SQL to read the look up table once   
 and store code / descriptions in local variables which can be accessed later many times   
  
Problem2:- simultaneous update/select operations on the same block   
  
Solutions:- modify the application to commit frequently and reasonbly   
 => CR clones need not be created   
  
  
Read Waits:-   
-----------   
  
DB File Scatter Read => waits for multiple blocks to read from disk while Full Table Scan   
  
Possible Solution:- try to cache frequently used small tables   
  
DB File Sequential Read => waits for single block read from disk while Index Full Scan   
  
possible solution:- check indexes on the table to ensure that the right index is being used   
 check the column order of the composite index with WHERE Clause   
  
storage disks   
- hard disks   
- flash disks (available by default in Exadata Server)   
  
  
Tuning PGA:-   
-----------   
  
What are the other memory structures in PGA?   
 - Hash Area (hash join operations for the SELECT)   
 - SORT Area (sort process in case or ORDER BY, GROUP BY, DISTINCT, etc.)   
 - workarea\_size\_policy parameter   
 - AUTO (default = HASH\_AREA and SORT\_AREA automatically set)   
 - by setting PGA\_AGGREGATE\_TARGET   
 - MANUAL   
  
what is workarea in PGA?   
- a memory area for processing   
  
How to set the optimal size for PGA?   
- based on information about optimal pass, one pass and multi pass operations   
  
Examples:-   
  
1) assume workarea size is 50 MB   
 SELECT require 25MB for processing   
 25MB < 50 MB => optimal pass (very good)   
  
  
2) assume workarea size is 50MB   
 SELECT require 75MB for processing   
50MB + 25 MB = 75 MB => one pass (ok)   
  
  
3) assume workarea size is 50MB

SELECT require 125 MB for processing   
50MB + 50MB + 25MB => more than one pass => multi pass (very bad)   
  
sga\_target <= sga\_max\_size   
  
memory\_target <= memory\_max\_target   
  
sga\_target = 500 MB => within in SGA + pga\_aggregate\_target   
  
memory\_target => SGA + PGA

DB Time is computed by adding the CPU time and wait time of all sessions (excluding the waits for idle events)  
  
**An AWR report shows the total DB Time for the instance**

(in the section “Time Model System Stats”) during the period covered by the AWR snapshots. If the time model  
statistic DB CPU consumes most of the database time for the instance, it shows the database was actively processing most of the time.DB time, which represents activity on your database. If the DB time exceeds the elapsed time, it denotes a busy database. If it is a lot higher than the elapsed time, it may mean that some sessions are waiting for resources  
  
  
SQL script for getting AWR Report on RAC database:  
SQL>@$ORACLE\_HOME/rdbms/admin/awrgrpt.sql  
  
SQL script for getting AWR Report for  single instance:  
SQL>@$ORACLE\_HOME/rdbms/admin/awrrpt.sql  
  
SQL script for getting ASH Report on RAC database:  
SQL>@$ORACLE\_HOME/rdbms/admin/ashrpti.sql  
  
SQL script for getting ASH Report for single Instance:  
SQL>@$ORACLE\_HOME/rdbms/admin/ashrpt.sql  
  
Comparing Database Performance Between Two Periods  
The AWR Compare Periods report compares performance between two different time periods, with each time  
period involving a different pair of snapshots.   
  
 @$ORACLE\_HOME/rdbms/admin/awrddrpt.sql  
  
  
  
  
Query Execution Phases (Parse, Bind, Execute and Fetch)   
  
Parse Phase   
 - two checks   
 - syntax check   
 Ex:- select \* fromm emp; ----->"from" keyword NOT found   
 select \* emp;   
  
- symantic check   
 Ex: select \* from emp; (assume, there is NO table with the ename "emp")   
 =>"Table or View does not exists"  
 update scott.emp set sal=sal+1000; (connected to "hr" schema)   
 =>"Insufficient Privileges"  
 select name from emp; (assume, here "name" NOT found in the "emp")   
 =>"Invalid identifier "name""  
  
  
- two types of parses   
 - soft parse   
 - hard parse  
  
-- Hard Parse  
If a new SQL statement is issued which does not exist in the shared pool then this has to be parsed fully.   
Eg: Oracle has to allocate memory for the statement from the shared pool, check the statement syntactically   
and semantically etc... This is referred to as a hard parse and is very expensive in both terms of CPU used   
and in the number of latch gets performed.   
  
--Soft Parse  
If a session issues a SQL statement which is already in the shared pool AND it can use an existing version   
of that statement then this is known as a 'soft parse'.   
As far as the application is concerned it has asked to parse the statement.   
  
if two statements are textually identical but cannot be shared then these are called 'versions' of the same statement.   
If Oracle matches to a statement with many versions it has to check each version in turn to see   
if it is truely identical to the statement currently being parsed. Hence high version counts are best avoided.   
  
shared pool   
 - library cache   
 - shared SQL Area   
 - holds SQL CURSOR(s)   
 - data dictionary cache   
 - result cache   
  
  
SQL CURSOR?   
 - workarea   
- memory area   
 - for workload of a particular SQL Statement   
 - information which is required to run a particular SQL St.   
 - statistics, selectivity, cardinality, explain plan, access path,   
 optimizer join method, etc.

- when you issue a particular SQL St. first time, then CURSOR creates in shared SQL Area   
  
  
  
Examples:-   
  
    first time:   SELECT \* FROM emp;    => store in library cache (Hard Parse)   
        2nd time: SELECT \* FROM emp;    => soft parse   
        3rd time: select \* from emp;    => Hard Parse   
        4th time: SELECT \*  FROM emp;    => Hard Parse   
        5th time: SELECT \* FROM emp WHERE deptno=10;    => HP   
  
CURSOR\_SHARING=EXACT (default)    => character by character, space by space checking   
  
      
    select \* from emp where deptno=10;    => hard parse   
    select \* from emp where deptno=20;    => hard parse   
    select \* from emp where deptno=30;    => hard parse   
    select \* from emp where deptno=40;    => hard parse   
  
  
How to avoid hard parses?   
  
a) follow standards   
b) use bind variables in place of literals   
  
    ex: select \* from emp where deptno = :dno;    (var command in sqlplus)   
  
  
  
CURSOR\_SHARING   
    - EXACT (default)    (good for OLTP) >90% DML operations   
    - SIMILAR (deprecated from 12c)   
    - FORCE   
  
  
CURSOR\_SHARING=FORCE    (good for OLAP) >90% are SELECT operations   
  
    select \* from emp where deptno=10;    => hard parse   
    select \* from emp where deptno=20;    => soft parse   
    select \* from emp where deptno=30;    => soft parse   
    select \* from emp where deptno=40;    => soft parse   
  
  
      
    select \* from emp where deptno = :SYS.bind\_variable;   
  
  
  
  
Bind Phase   
    - optional   
    - useful in case of any bind variable in the query   
      
  
  
every SQL query identified by SQL ID   
every execution plan of a Query identified by Plan Hash Value   
can I have same SQL ID with different Plan Hash Values?   
Ans:  YES   
  
  
  
select \* from emp where deptno = :dno;   
  
:dno=10    => sqlid1, plan1   
:dno=20 => sqlid1, plan2   
:dno=30 => sqlid1, plan3   
  
  
the above is possible from 11g R2 due to Adaptive Cursor Sharing   
  
statistics is important input to Optimizer to run a query   
from 11g R2, selectivity is also important to optimizer   
  
assume, emp has 100 rows, 10th dept has 15 rows, 20th dept has 65 rows, 30th has 20 rows   
  
  
selectivity (10th Dept) = 15/100 = 0.15   
selectivity (20th dept) = 65/100 = 0.65   
selectivity (30th dept) = 20/100 = 0.2   
  
  
select \* from emp where deptno = :dno;   
  
:dno=10    => 0.15, plan1   
:dno=20 => 0.65, plan2   
:dno=30 => 0.2, plan1   
  
  
Bind Peaking   
  
  
  
select \* from emp where deptno = :dno;   
  
:dno=10    => 0.15, plan1   
:dno=20 => 0.65, plan1   
:dno=30 => 0.2, plan1

### Advance Database Performance Tools

1) RDA Instalation and its use

5) OS watcher

6) One case Study of performance tuning .

7) Various option of sql tracing & tkprof

8) Various option for explain plan generation

9) Test case of your analysis of SQLT, AWR report, explain plan analysis

10) Various OS commands for identify bottleneck

1) **Remote Diagnostics Agent(RDA)**

RDA is a utility a set of shell scripts or a PERL script, that can be downloaded from Oracle Support to collect diagnostics information for an Oracle database and it's environment(RAC, ASM, Exadata).

This utility is focused at collecting information that will aid in program diagnosis when logging a call, Oracle support will often request that we install the RDA utility, run it and upload the output to Oracle Support for analysis.

It’s not only a great tool for troubleshooting but also very helpful for documenting an Oracle environment.  
RDA offers lots of reporting options and provides easy to read results. You can run it on just about any version of the Database or Oracle Applications or Operating System and it is smart enough to figure out where to go and what to gather.  
  
Once It is installed and run rda.sh or rda.pl, you have to answer some questions and send it off to gather information about your environment. As result you will get a lot of TXT and HTML files.

**RDA Installation**

Download the patch from Metalink, FTP to database box and unzip it.

oradb@KEMGSADEDBOT01 tmp]$ unzip p21769913\_8111638\_Linux-x86-64.zip

Archive: p21769913\_8111638\_Linux-x86-64.zip

inflating: readme.txt

creating: rda/

creating: rda/Convert/

creating: rda/Convert/Common/

creating: rda/Convert/DB/

[oracle@lab rda]$ ls

admin DISCLAIM.txt mesg RDA README\_irda.txt sdboot.pl tools

collect engine modules rda.cmd README\_Unix.txt sdci.cmd

Convert hcve output rda.com README\_upgrade.txt sdci.pl

da IRDA output.cfg rda.pl README\_VMS.txt sdci.sh

dfw irda.pl Pod rda.sh README\_Windows.txt temp

[oracle@lab **rda]$**

**Run the RDA command**

[oracle@lab rda]$ sh rda.sh

------------------------------------------------------------------------------

RDA Data Collection Started 11-Jun-2016 00:57:32

------------------------------------------------------------------------------

Processing RDA.BEGIN module ...

Enter the password for "SYSTEM" at "cdb1":

Re-enter it to confirm:

Processing OS.PERF module ...

Processing RDA.CONFIG module ...

Processing SAMPLE.SAMPLE module ...

Processing OS.OS module …

Processing OS.PROF module ...

Processing OS.NET module ...

Processing OS.INST module ...

Processing DB.DBA module ...

Processing OFM.WREQ module ...

This can take time. Be patient ...

Processing OFM.IREQ module ...

Processing RDA.LOAD module ...

Processing RDA.END module ...

------------------------------------------------------------------------------

RDA Data Collection Ended 11-Jun-2016 00:58:26

**[oracle@lab rda]$ ./rda.sh -h**

Usage: rda.pl [-bcdfilnvwxy] [-ABCDEGHIKLMPQRSTV] [-e edt] [-m dir] [-o out]

[-p nam] [-s nam] [-t lvl] arg ...

-A Authentify user through the result set definition file

-B Start background collection

-C Collect diagnostic information

-D Delete specified modules from the collection

-E Explain specified error numbers

-G Convert report files to XML format

-H Halt background collection

-I Regenerate the index

-K Kill background collection

-L List the available modules, profiles, and conversion groups

-M Display the related manual pages

-O Render output specifications from the standard input

-P Package the reports (tar or zip)

-Q Display the related setup questions

-R Generate specified reports

-S Setup specified modules

-T Execute test modules

-V Display component version numbers

-b Do not backup result set definition file before saving

-c Check the RDA installation and exit

-d Set debug mode

-e edt Specify a list of alternate setting definitions (var=val,...)

-f Set force mode

-g grp Specify the XML conversion group

-h Display the command usage and exit

-i Read settings from the standard input

-l Use a lock file to prevent concurrent usage of a definition file

-m dir Specify the module directory ('collect' by default)

-n Start a new data collection

-o out Specify the file for background collection output redirection

-p nam Specify the collection profile ('Default' by default)

-q Set quiet mode

-s nam Specify the result set name ('output' by default)

-t lvl Specify the trace level

-u lng Specify the language and character set for user interactions

-v Set verbose mode

-w Wait as long as the background collection daemon is active

-x Produce cross references

-y Accept all defaults and skip all pauses

[oracle@ibmlab rda]$

RDA Output

[oracle@lab rda]$ ls -ltr

total 5236

-r-xr-xr-x 1 oracle oinstall 9489 Mar 7 17:48 sdci.sh

-r-xr-xr-x 1 oracle oinstall 26981 Mar 7 17:48 sdci.pl

-r-xr-xr-x 1 oracle oinstall 8903 Mar 7 17:48 sdci.cmd

-r-xr-xr-x 1 oracle oinstall 25564 Mar 7 17:48 sdboot.pl

-r--r--r-- 1 oracle oinstall 16172 Mar 7 17:48 README\_Windows.txt

-r--r--r-- 1 oracle oinstall 17555 Mar 7 17:48 README\_VMS.txt

-r--r--r-- 1 oracle oinstall 5144 Mar 7 17:48 README\_upgrade.txt

-r--r--r-- 1 oracle oinstall 16262 Mar 7 17:48 README\_Unix.txt

-r--r--r-- 1 oracle oinstall 3346 Mar 7 17:48 README\_irda.txt

-r-xr-xr-x 1 oracle oinstall 11698 Mar 7 17:48 rda.sh

-r-xr-xr-x 1 oracle oinstall 43807 Mar 7 17:48 rda.pl

-r-xr-xr-x 1 oracle oinstall 7754 Mar 7 17:48 rda.com

-r-xr-xr-x 1 oracle oinstall 9298 Mar 7 17:48 rda.cmd

-r-xr-xr-x 1 oracle oinstall 14688 Mar 7 17:48 irda.pl

-r--r--r-- 1 oracle oinstall 4106 Mar 7 17:48 DISCLAIM.txt

drwxr-xr-x 2 oracle oinstall 4096 Mar 8 18:13 tools

drwxr-xr-x 19 oracle oinstall 4096 Mar 8 18:13 RDA

drwxr-xr-x 3 oracle oinstall 4096 Mar 8 18:13 Pod

drwxr-xr-x 2 oracle oinstall 4096 Mar 8 18:13 modules

drwxr-xr-x 6 oracle oinstall 4096 Mar 8 18:13 mesg

drwxr-xr-x 3 oracle oinstall 4096 Mar 8 18:13 IRDA

drwxr-xr-x 9 oracle oinstall 4096 Mar 8 18:13 hcve

drwxr-xr-x 2 oracle oinstall 4096 Mar 8 18:13 engine

drwxr-xr-x 3 oracle oinstall 4096 Mar 8 18:13 dfw

drwxr-xr-x 8 oracle oinstall 4096 Mar 8 18:13 Convert

drwxr-xr-x 15 oracle oinstall 4096 Mar 8 18:13 collect

drwxr-xr-x 5 oracle oinstall 4096 Mar 8 18:13 admin

drwxr-xr-x 7 oracle oinstall 4096 Mar 8 18:13 da

drwxr-x--- 6 oracle oinstall 4096 Jun 11 01:03 temp

-rw-r----- 1 oracle oinstall 149989 Jun 11 01:03 set.bak

-rw-r----- 1 oracle oinstall 148687 Jun 11 01:05 set.cfg

drwxr-x--- 5 oracle oinstall 4096 Jun 11 01:05 set

-rw-r----- 1 oracle oinstall 2095445 Jun 11 01:05 RDA\_set\_ibmlab.zip

-rw-r----- 1 oracle oinstall 148490 Jun 11 01:10 output.bak

-rw-r--r-- 1 oracle oinstall 57998 Jun 11 01:10 output.txt

-rw-r----- 1 oracle oinstall 148669 Jun 11 01:14 output.cfg

drwxr-x--- 5 oracle oinstall 4096 Jun 11 01:14 output

**-rw-r----- 1 oracle oinstall 2094613 Jun 11 01:14 RDA\_output\_ibmlab.zip**

**[oracle@ibmlab rda]$**

### 2: OS Watcher Installation Steps:-

OS Watcher is a series of shell scripts that collect specific kinds of data, using operating system diagnostic utilities. Control is passed to individually spawned operating system data collector processes, which in turn collect specific data, time stamp the data output, and append the data to pre-generated and named files. Each data collector will have its own file, created and named by the File Manager process. OSW invokes the distinct operating system utilities listed below as data collectors. OSW will not put any significant performance affecting load on the system. It will have the same impact as running the regular OS command like netstat, ps etc. These utilities will be supported, or their equivalents, as available for each supported target platform:

\* ps

\* top

\* mpstat

\* iostat

\* netstat

\* traceroute

\* vmstat

The size of the archived files saved during the running of the OSW will be based on the user parameters set at the starting of OSW and the OS information. For example, if each file will contain an hour of data and the collection interval is 10 sec the amount of data will be bigger compared to collecting with an interval of 60 sec.

It is highly recommended that OSW be installed and run continuously on ALL cluster nodes, at all times Document 301137.1

Be sure to use separate directories per node for storing OSW output. When using OS Watcher in a RAC environment, each node must write its output files to a separate archive directory. Combining the output files under one archive (on shared storage) is not supported and causes the OSW tool to crash. Shared storage is fine, but each node needs a separate archive directory.

Document 301137.1 - OS Watcher User Guide

OSW for WINDOWS: OS Watcher for Windows is no longer supported. It has been replace by the Cluster Health Monitor. Please see Document ID: 736752.1 for more information, and how to download, the Cluster Health Monitor.

[oracle@ibmlab jb]$ pwd

/home/oracle/jb

[oracle@ibmlab jb]$ ls

oswbb733.tar

[oracle@ibmlab jb]$ tar -xvf oswbb733.tar

oswbb/

oswbb/docs/

oswbb/docs/The\_Analyzer/

oswbb/docs/The\_Analyzer/OSWatcherAnalyzerOverview.pdf

oswbb/docs/The\_Analyzer/oswbbaUserGuide.pdf

oswbb/docs/The\_Analyzer/oswbba\_README.txt

oswbb/docs/OSWatcher/

oswbb/docs/OSWatcher/oswbb\_README.txt

oswbb/docs/OSWatcher/OSWatcherUserGuide.pdf

oswbb/Exampleprivate.net

oswbb/nfssub.sh

oswbb/stopOSWbb.sh

oswbb/call\_du.sh

oswbb/iosub.sh

oswbb/OSWatcherFM.sh

oswbb/ifconfigsub.sh

oswbb/ltop.sh

oswbb/mpsub.sh

oswbb/call\_uptime.sh

oswbb/psmemsub.sh

oswbb/tar\_up\_partial\_archive.sh

oswbb/oswnet.sh

oswbb/vmsub.sh

oswbb/call\_sar.sh

oswbb/oswib.sh

oswbb/startOSWbb.sh

oswbb/Example\_extras.txt

oswbb/oswsub.sh

oswbb/oswbba.jar

oswbb/OSWatcher.sh

oswbb/tarupfiles.sh

oswbb/xtop.sh

oswbb/src/

oswbb/src/Thumbs.db

oswbb/src/OSW\_profile.htm

oswbb/src/tombody.gif

oswbb/src/missing\_graphic.gif

oswbb/src/coe\_logo.gif

oswbb/src/watch.gif

oswbb/src/oswbba\_input.txt

oswbb/oswrds.sh

[oracle@ibmlab jb]$

**Start OS watcher using the scripts as oracle user to capture data every 15 second:**

nohup ./startOSWbb.sh 15 300 gzip &

[oracle@ibmlab oswbb]$ ps -ef |grep -i osw

oracle 21464 1 0 19:39 pts/1 00:00:00 /bin/sh ./OSWatcher.sh 15 300 gzip

oracle 21522 21464 0 19:39 pts/1 00:00:00 /bin/sh ./OSWatcherFM.sh 300 /home/oracle/jb/oswbb/archive

oracle 21943 21266 0 19:41 pts/1 00:00:00 grep -i osw

[oracle@ibmlab oswbb]$

**To stop OSWatcher:**

[oracle@ibmlab oswbb]$ ./stopOSWbb.sh

[oracle@ibmlab oswbb]$

**To uninstall the OSWatche:**

simply delete the oswatcher parent directory.

### 6) Three case Study of performance tuning

**Case 1.**

I got a call from the end user claiming that the user is not able to process the transaction. I have followed the below process to fix the issue.

I have logged on to the server and checked the user transaction information and status of the user and was looking for the locks on the database.

To show the information, I will take the scott schema as the example and explain the same. We need to log in to the database as two users, both would be modifying the same record which makes the lock on the record before it gets committed.

SQL> update emp set sal=1000 where deptno=10;

3 rows updated.

SQL> show user;

USER is "SCOTT"

SQL>

open the another session and try to modify the same record. Which will not complete the transaction as it's going on lock.

SQL> show user

USER is "SCOTT"

SQL> update emp set sal=1000 where deptno=10;

Use below query to find out who is blocking whoom.

SQL> select l1.sid, ' IS BLOCKING ', l2.sid

from v$lock l1, v$lock l2

where l1.block =1 and l2.request > 0

and l1.id1=l2.id1

and l1.id2=l2.id2; 2 3 4 5

SID 'ISBLOCKING' SID

---------- ------------- ----------

37 IS BLOCKING 39

SQL>

Using below query we can find out the user info and the query which is blocking.

SQL> Select sid,program,action,username from v$session where sid in(37,39);

SID PROGRAM

---------- ------------------------------------------------

ACTION

----------------------------------------------------------------

USERNAME

------------------------------

37 sqlplus@ibmlab.localdomain (TNS V1-V3)

SCOTT

39 sqlplus@ibmlab.localdomain (TNS V1-V3)

SCOTT

SID PROGRAM

---------- ------------------------------------------------

ACTION

----------------------------------------------------------------

USERNAME

------------------------------

SQL> Select sql\_text from v$sql s, v$session sess where s.sql\_id=sess.sql\_id and sess.sid=37;

no rows selected

SQL> Select sql\_text from v$sql s, v$session sess where s.sql\_id=sess.sql\_id and sess.sid=39;

SQL\_TEXT

--------------------------------------------------------------------------------

update emp set sal=1000 where deptno=10

SQL>

We can kill the 39 session to release the locking or we can request the user to go ahead and finish the commit. Most of the cases we need to kill the session after taking the approval from the customer. We will kill the blocking session by using session sid.

SQL> select sid,serial# from v$session where sid='37';

SID SERIAL#

---------- ----------

37 12144

SQL> alter system kill session '37,12144';

System altered.

SQL>

The movement we kill the session. Locks will be over the end user transaction will complete. Please see the below log of the end user session.

SQL> show user

USER is "SCOTT"

SQL> update emp set sal=1000 where deptno=10;

3 rows updated.

SQL>

**Case 2**

1) User complained about sudden poor performance of user job

Checked whether job was running or not

$ps -ef|grep -i udmd4090

oracle 44368126 43122766 0 15:41:48 pts/0 0:00 grep -i udmd4090

bprd001 10158450 1 0 15:40:40 - 0:00 /bin/ksh /prod/dmprod/jobs\_exec/udmd4090.sh

and check database level performance issue and blocking session as well

--To check blocking session ---

SELECT DECODE(request,0,'Holder: ','Waiter: ') ||

sid sess, id1, id2, lmode, request, type

FROM V$LOCK

WHERE (id1, id2, type) IN (SELECT id1, id2, type FROM V$LOCK WHERE request > 0)

ORDER BY id1, request;

SQL> select count(1),event from v$session group by event;

COUNT(1) EVENT

---------- ----------------------------------------------------------------

10 PL/SQL lock timer

228 SQL\*Net message from client

1 SQL\*Net message from dblink

1 SQL\*Net message to client

1 Streams AQ: qmn coordinator idle wait

1 Streams AQ: qmn slave idle wait

1 Streams AQ: waiting for time management or cleanup tasks

1 TCP Socket (KGAS)

1 db file scattered read

1 db file sequential read

2 pipe get

COUNT(1) EVENT

---------- ----------------------------------------------------------------

1 pmon timer

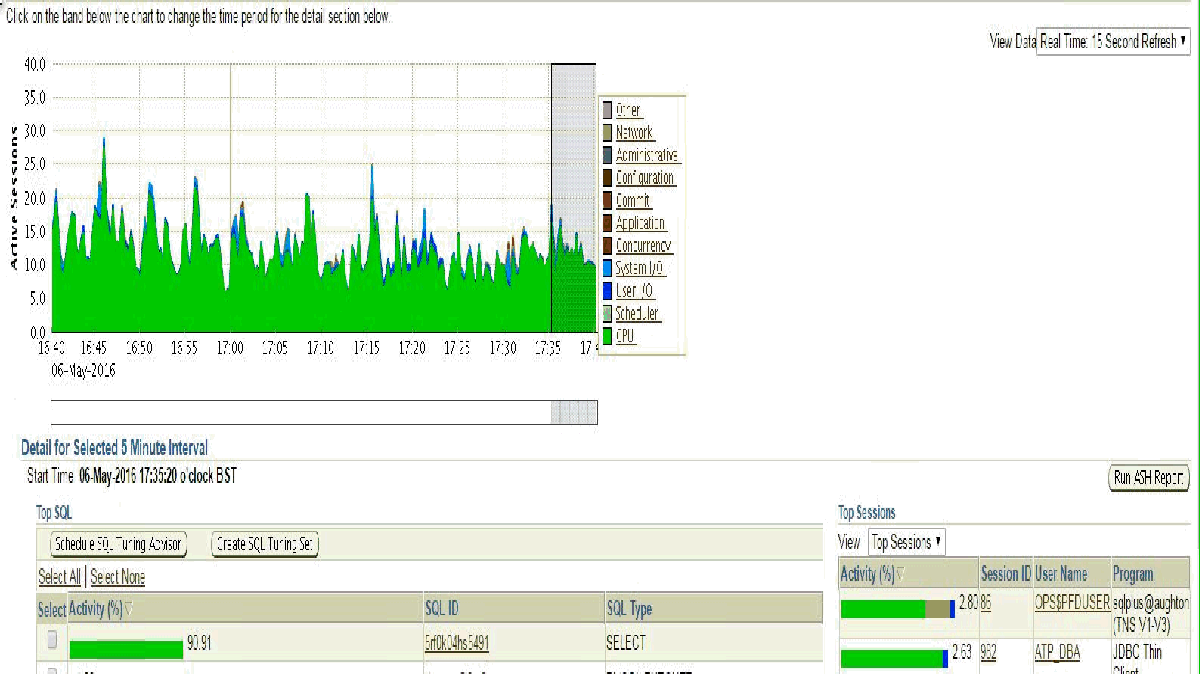
14 rdbms ipc message

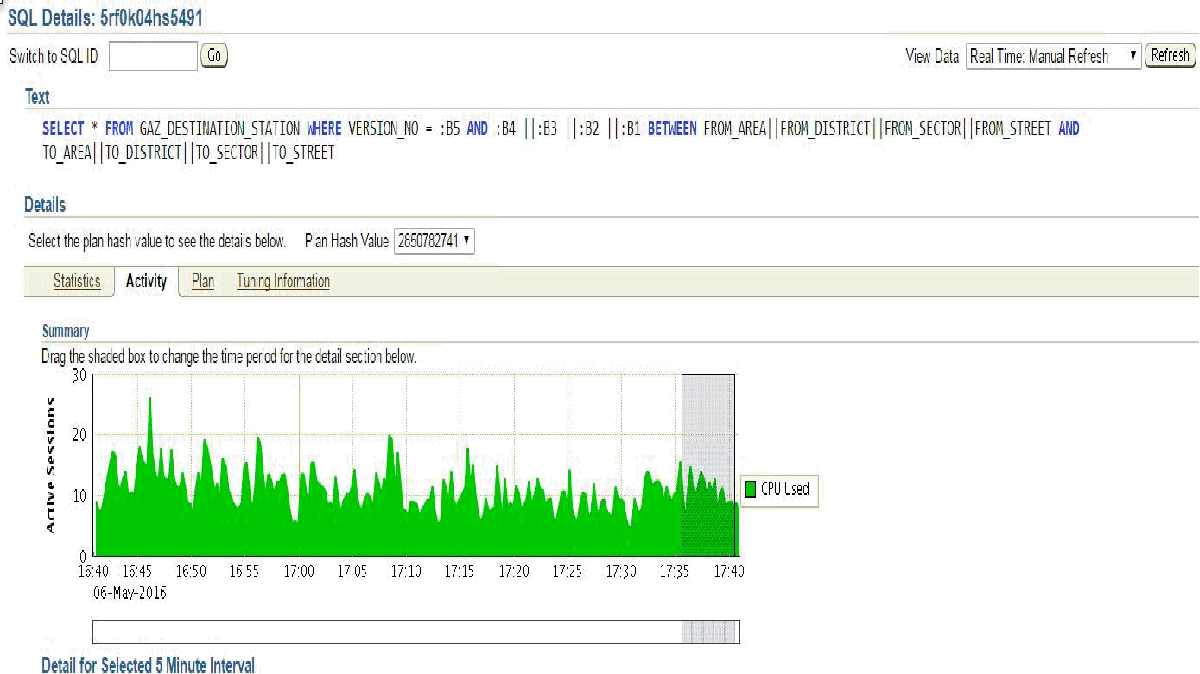
1 smon timer

but did not found any major performance degradation database level

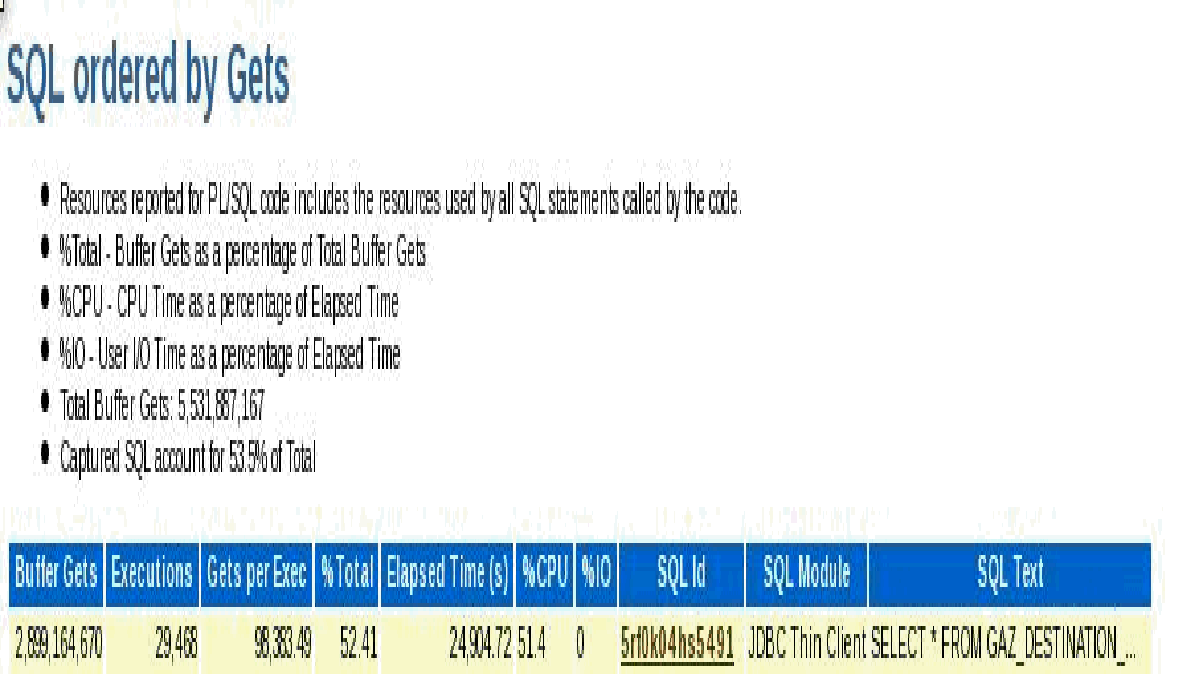
Therefore for further analysis ,check from GRID and generated AWR and ASH and ADDM report for same time period when performance issue reported an

**Grid output**

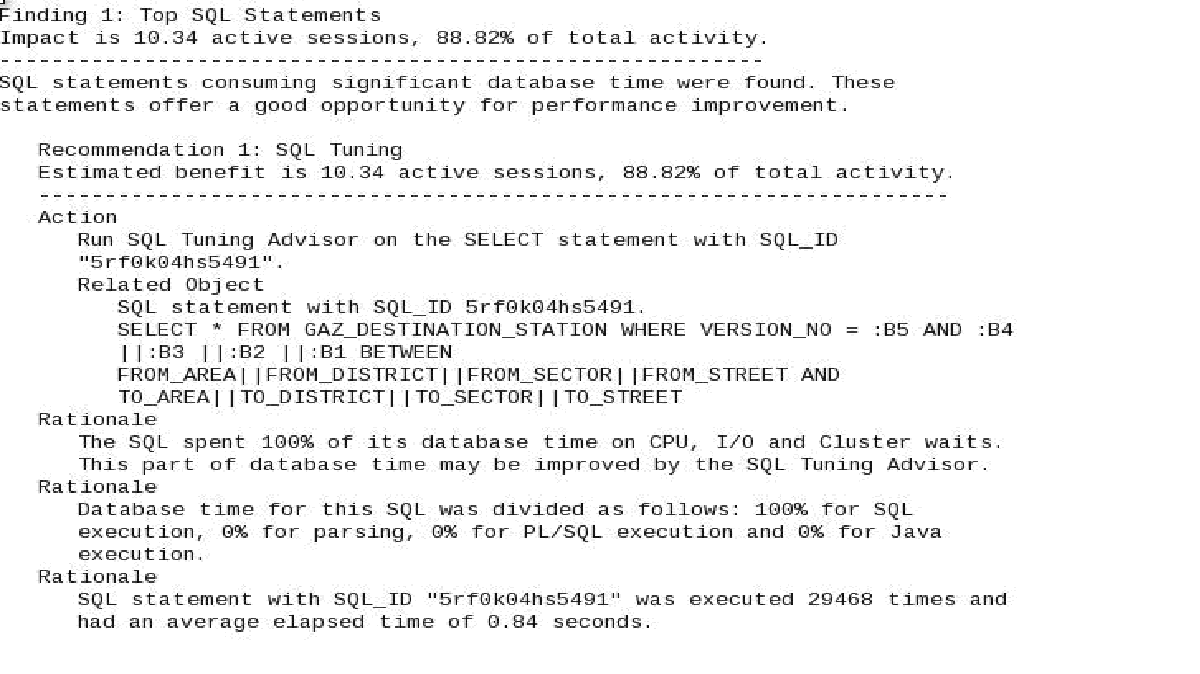




**Output from AWR**

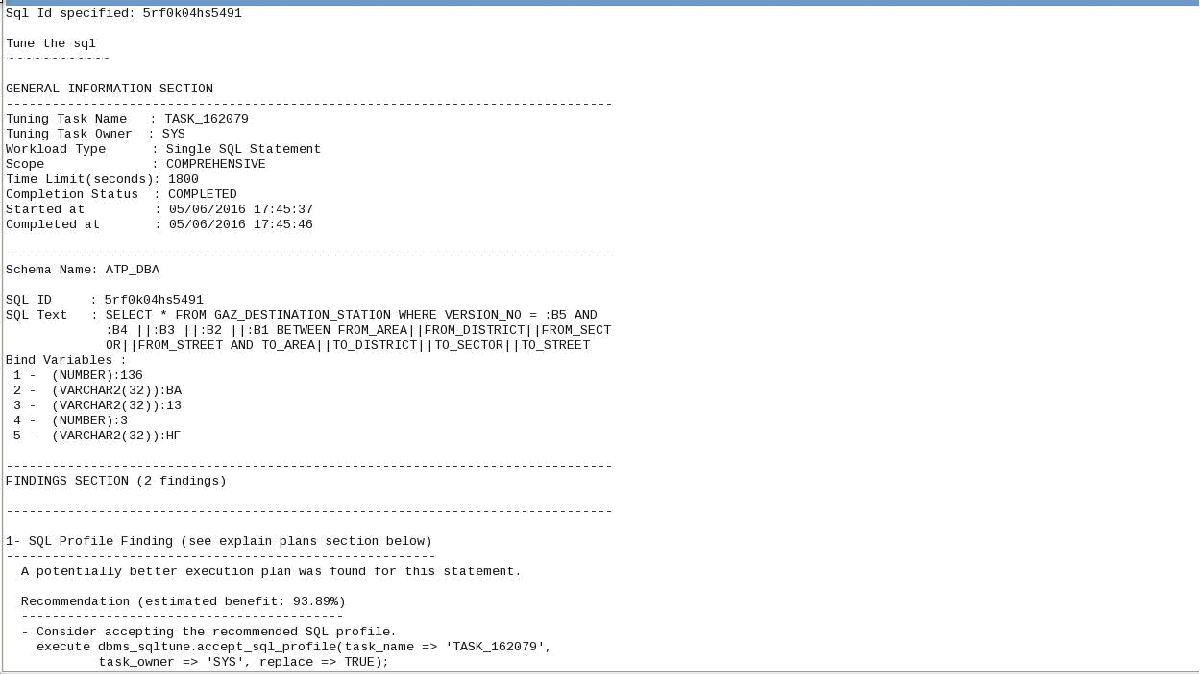


**Output from ADDM report**



**SQLT recommendation**

upon analysis ,We observed the creating and setting profile will help optimizer to choose correct plan and generated SQLT report as below

CREATE INDEX ATP\_DBA.GA\_DE\_ST\_IN\_02 ON ATP\_DBA.GAZ\_DESTINATION\_STATION

(FROM\_AREA||FROM\_DISTRICT||from\_sector||FROM\_STREET)

TABLESPACE TSPACEI01;

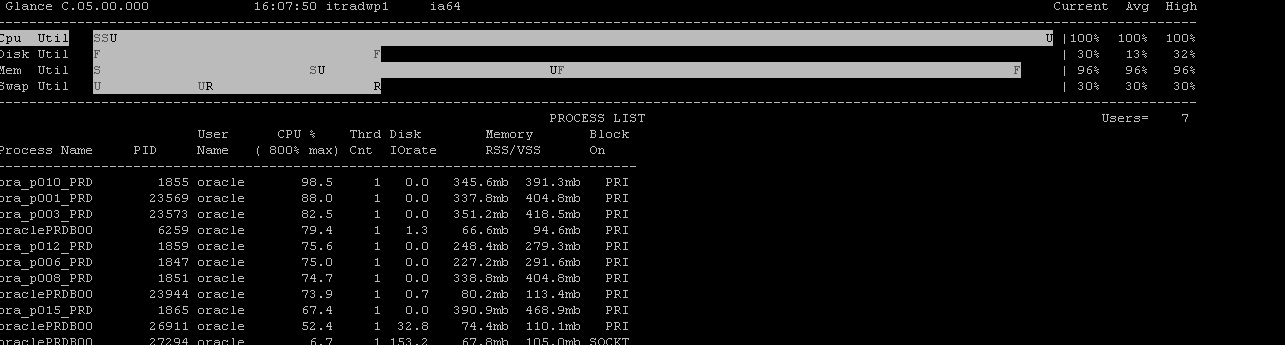
CREATE INDEX ATP\_DBA.GA\_DE\_ST\_IN\_03 ON ATP\_DBA.GAZ\_DESTINATION\_STATION

(TO\_AREA||TO\_DISTRICT||to\_sector||TO\_STREET)

TABLESPACE TSPACEI01;

**Third case:**

**The Below Script is causing database performance degradation. Please see the same sqlid 4p0xrmd31t34c** is running by multiple session. Those session is coming in Top session in GLANCE output



**$ ps -ef|grep 1855**

oracle 8093 27204 0 16:09:26 pts/7 0:00 grep 1855

oracle 1855 1 255 15:48:08 ? 17:12 ora\_p010\_PRDB0073

**$ ps -ef|grep 23569**

oracle 23569 1 252 15:24:21 ? 9:47 ora\_p001\_PRDB0073

oracle 8134 27204 0 16:09:52 pts/7 0:00 grep 23569

**$ ps -ef|grep 1859**

oracle 8510 27204 0 16:10:31 pts/7 0:00 grep 1859

oracle 1859 1 255 15:48:08 ? 18:11 ora\_p012\_PRDB0073

$ ps -ef|grep 1847

oracle 1847 1 16 15:48:08 ? 3:17 ora\_p006\_PRDB0073

oracle 8521 27204 0 16:10:39 pts/7 0:00 grep 1847

**$ ps -ef|grep 1851**

oracle 1851 1 254 15:48:08 ? 6:29 ora\_p008\_PRDB0073

oracle 8540 27204 0 16:10:49 pts/7 0:00 grep 1851

**$ ps -ef|grep 1865**

oracle 8599 27204 1 16:11:09 pts/7 0:00 grep 1865

oracle 1865 1 0 15:48:08 ? 10:12 ora\_p015\_PRDB0073

$

Enter value for 1: 1855

old 2: where p.addr=s.paddr and p.spid=&1

new 2: where p.addr=s.paddr and p.spid=1855

SQL\_ID SQL\_HASH\_VALUE PREV\_HASH\_VALUE SID SERIAL# USERNAME EVENT TO\_CHAR(S.LOGON\_TIME,

------------- -------------- --------------- ---------- ---------- ------------------------------ ---------------------------------------------------------------- ---------------------

**4p0xrmd31t34c** 1176276108 0 145 18870 WHS\_VIEWER direct path write temp 23-may:15:48

SQL> /

Enter value for 1: 23569

old 2: where p.addr=s.paddr and p.spid=&1

new 2: where p.addr=s.paddr and p.spid=23569

SQL\_ID SQL\_HASH\_VALUE PREV\_HASH\_VALUE SID SERIAL# USERNAME EVENT TO\_CHAR(S.LOGON\_TIME,

------------- -------------- --------------- ---------- ---------- ------------------------------ ---------------------------------------------------------------- ---------------------

**4p0xrmd31t34c** 1176276108 0 301 17086 WHS\_VIEWER PX Deq: Table Q Normal 23-may:15:48

SQL> /

Enter value for 1: 23573

old 2: where p.addr=s.paddr and p.spid=&1

new 2: where p.addr=s.paddr and p.spid=23573

SQL\_ID SQL\_HASH\_VALUE PREV\_HASH\_VALUE SID SERIAL# USERNAME EVENT TO\_CHAR(S.LOGON\_TIME,

------------- -------------- --------------- ---------- ---------- ------------------------------ ---------------------------------------------------------------- ---------------------

**4p0xrmd31t34c** 1176276108 0 245 1984 WHS\_VIEWER PX Deq: Table Q Normal 23-may:15:48

SQL> /

Enter value for 1: 1859

old 2: where p.addr=s.paddr and p.spid=&1

new 2: where p.addr=s.paddr and p.spid=1859

SQL\_ID SQL\_HASH\_VALUE PREV\_HASH\_VALUE SID SERIAL# USERNAME EVENT TO\_CHAR(S.LOGON\_TIME,

------------- -------------- --------------- ---------- ---------- ------------------------------ ---------------------------------------------------------------- ---------------------

**4p0xrmd31t34c** 1176276108 0 227 42381 WHS\_VIEWER direct path read temp 23-may:15:48

SQL> /

Enter value for 1: 1847

old 2: where p.addr=s.paddr and p.spid=&1

new 2: where p.addr=s.paddr and p.spid=1847

SQL\_ID SQL\_HASH\_VALUE PREV\_HASH\_VALUE SID SERIAL# USERNAME EVENT TO\_CHAR(S.LOGON\_TIME,

------------- -------------- --------------- ---------- ---------- ------------------------------ ---------------------------------------------------------------- ---------------------

**4p0xrmd31t34c** 1176276108 0 248 8364 WHS\_VIEWER PX Deq: Table Q Normal 23-may:15:48

SQL> /

Enter value for 1: 1851

old 2: where p.addr=s.paddr and p.spid=&1

new 2: where p.addr=s.paddr and p.spid=1851

SQL\_ID SQL\_HASH\_VALUE PREV\_HASH\_VALUE SID SERIAL# USERNAME EVENT TO\_CHAR(S.LOGON\_TIME,

------------- -------------- --------------- ---------- ---------- ------------------------------ ---------------------------------------------------------------- ---------------------

**4p0xrmd31t34c** 1176276108 0 273 7054 WHS\_VIEWER direct path write temp 23-may:15:48

SQL> /

Enter value for 1: 1865

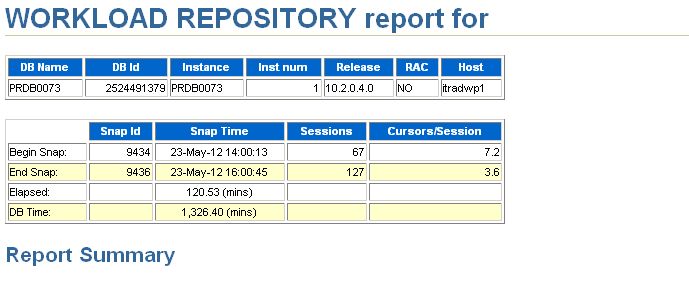
old 2: where p.addr=s.paddr and p.spid=&1

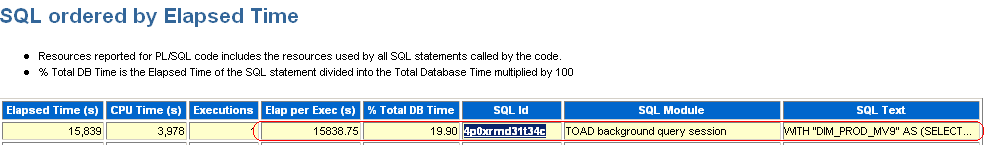
new 2: where p.addr=s.paddr and p.spid=1865

SQL\_ID SQL\_HASH\_VALUE PREV\_HASH\_VALUE SID SERIAL# USERNAME EVENT TO\_CHAR(S.LOGON\_TIME,

------------- -------------- --------------- ---------- ---------- ------------------------------ ---------------------------------------------------------------- ---------------------

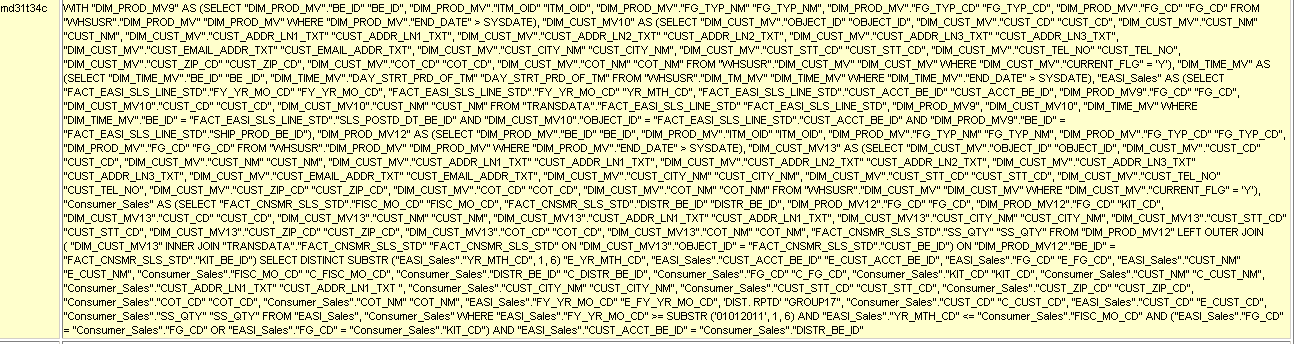
**4p0xrmd31t34c** 1176276108 0 406 26464 WHS\_VIEWER PX Deq: Table Q Normal 23-may:15:48







**Complete Query**



**ADDM Recommendation**

DETAILED ADDM REPORT FOR TASK 'TASK\_8538' WITH ID 8538

------------------------------------------------------

Analysis Period: 23-MAY-2012 from 15:00:29 to 16:00:45

Database ID/Instance: 2524491379/1

Database/Instance Names: PRDB0073/PRDB0073

Host Name: itradwp1

Database Version: 10.2.0.4.0

Snapshot Range: from 9435 to 9436

Database Time: 47551 seconds

Average Database Load: 13.2 active sessions

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

FINDING 1: 63% impact (30173 seconds)

-------------------------------------

SQL statements consuming significant database time were found.

RECOMMENDATION 1: SQL Tuning, 33% benefit (15839 seconds)

ACTION: Investigate the SQL statement with SQL\_ID "4p0xrmd31t34c" for

possible performance improvements.

RELEVANT OBJECT: SQL statement with SQL\_ID 4p0xrmd31t34c and

PLAN\_HASH 2824768675

WITH "DIM\_PROD\_MV9" AS (SELECT "DIM\_PROD\_MV"."BE\_ID""BE\_ID",

"DIM\_PROD\_MV"."ITM\_OID""ITM\_OID",

"DIM\_PROD\_MV"."FG\_TYP\_NM""FG\_TYP\_NM",

"DIM\_PROD\_MV"."FG\_TYP\_CD""FG\_TYP\_CD",

"DIM\_PROD\_MV"."FG\_CD""FG\_CD"

FROM "WHSUSR"."DIM\_PROD\_MV""DIM\_PROD\_MV"

WHERE "DIM\_PROD\_MV"."END\_DATE"> SYSDATE),

"DIM\_CUST\_MV10"

AS (SELECT "DIM\_CUST\_MV"."OBJECT\_ID""OBJECT\_ID",

"DIM\_CUST\_MV"."CUST\_CD""CUST\_CD",

"DIM\_CUST\_MV"."CUST\_NM""CUST\_NM",

"DIM\_CUST\_MV"."CUST\_ADDR\_LN1\_TXT""CUST\_ADDR\_LN1\_TXT",

"DIM\_CUST\_MV"."CUST\_ADDR\_LN2\_TXT""CUST\_ADDR\_LN2\_TXT",

"DIM\_CUST\_MV"."CUST\_ADDR\_LN3\_TXT""CUST\_ADDR\_LN3\_TXT",

"DIM\_CUST\_MV"."CUST\_EMAIL\_ADDR\_TXT""CUST\_EMAIL\_ADDR\_TXT",

"DIM\_CUST\_MV"."CUST\_CITY\_NM""CUST\_CITY\_NM",

"DIM\_CUST\_MV"."CUST\_STT\_CD""CUST\_STT\_CD",

"DIM\_CUST\_MV"."CUST\_TEL\_NO""CUST\_TEL\_NO",

"DIM\_CUST\_MV"."CUST\_ZIP\_CD""CUST\_ZIP\_CD",

"DIM\_CUST\_MV"."COT\_CD""COT\_CD",

"DIM\_CUST\_MV"."COT\_NM""COT\_NM"

FROM "WHSUSR"."DIM\_CUST\_MV""DIM\_CUST\_MV"

WHERE "DIM\_CUST\_MV"."CURRENT\_FLG" = 'Y'),

"DIM\_TIME\_MV"

AS (SELECT "DIM\_TIME\_MV"."BE\_ID""BE\_ID",

"DIM\_TIME\_MV"."DAY\_STRT\_PRD\_OF\_TM""DAY\_STRT\_PRD\_OF\_TM"

FROM "WHSUSR"."DIM\_TM\_MV""DIM\_TIME\_MV"

WHERE "DIM\_TIME\_MV"."END\_DATE"> SYSDATE),

"EASI\_Sales"

AS (SELECT "FACT\_EASI\_SLS\_LINE\_STD"."FY\_YR\_MO\_CD""FY\_YR\_MO\_CD",

"FACT\_EASI\_SLS\_LINE\_STD"."FY\_YR\_MO\_CD""YR\_MTH\_CD",

"FACT\_EASI\_SLS\_LINE\_STD"."CUST\_ACCT\_BE\_ID"

"CUST\_ACCT\_BE\_ID",

"DIM\_PROD\_MV9"."FG\_CD""FG\_CD",

"DIM\_CUST\_MV10"."CUST\_CD""CUST\_CD",

"DIM\_CUST\_MV10"."CUST\_NM""CUST\_NM"

FROM "TRANSDATA"."FACT\_EASI\_SLS\_LINE\_STD""FACT\_EASI\_SLS\_LINE\_STD",

"DIM\_PROD\_MV9",

"DIM\_CUST\_MV10",

"DIM\_TIME\_MV"

WHERE "DIM\_TIME\_MV"."BE\_ID" =

"FACT\_EASI\_SLS\_LINE\_STD"."SLS\_POSTD\_DT\_BE\_ID"

AND "DIM\_CUST\_MV10"."OBJECT\_ID" =

"FACT\_EASI\_SLS\_LINE\_STD"."CUST\_ACCT\_BE\_ID"

AND "DIM\_PROD\_MV9"."BE\_ID" =

"FACT\_EASI\_SLS\_LINE\_STD"."SHIP\_PROD\_BE\_ID"),

"DIM\_PROD\_MV12" AS (SELECT "DIM\_PROD\_MV"."BE\_ID""BE\_ID",

"DIM\_PROD\_MV"."ITM\_OID""ITM\_OID",

"DIM\_PROD\_MV"."FG\_TYP\_NM""FG\_TYP\_NM",

"DIM\_PROD\_MV"."FG\_TYP\_CD""FG\_TYP\_CD",

"DIM\_PROD\_MV"."FG\_CD""FG\_CD"

FROM "WHSUSR"."DIM\_PROD\_MV""DIM\_PROD\_MV"

WHERE "DIM\_PROD\_MV"."END\_DATE"> SYSDATE),

"DIM\_CUST\_MV13"

AS (SELECT "DIM\_CUST\_MV"."OBJECT\_ID""OBJECT\_ID",

"DIM\_CUST\_MV"."CUST\_CD""CUST\_CD",

"DIM\_CUST\_MV"."CUST\_NM""CUST\_NM",

"DIM\_CUST\_MV"."CUST\_ADDR\_LN1\_TXT""CUST\_ADDR\_LN1\_TXT",

"DIM\_CUST\_MV"."CUST\_ADDR\_LN2\_TXT""CUST\_ADDR\_LN2\_TXT",

"DIM\_CUST\_MV"."CUST\_ADDR\_LN3\_TXT""CUST\_ADDR\_LN3\_TXT",

"DIM\_CUST\_MV"."CUST\_EMAIL\_ADDR\_TXT""CUST\_EMAIL\_ADDR\_TXT",

"DIM\_CUST\_MV"."CUST\_CITY\_NM""CUST\_CITY\_NM",

"DIM\_CUST\_MV"."CUST\_STT\_CD""CUST\_STT\_CD",

"DIM\_CUST\_MV"."CUST\_TEL\_NO""CUST\_TEL\_NO",

"DIM\_CUST\_MV"."CUST\_ZIP\_CD""CUST\_ZIP\_CD",

"DIM\_CUST\_MV"."COT\_CD""COT\_CD",

"DIM\_CUST\_MV"."COT\_NM""COT\_NM"

FROM "WHSUSR"."DIM\_CUST\_MV""DIM\_CUST\_MV"

WHERE "DIM\_CUST\_MV"."CURRENT\_FLG" = 'Y'),

"Consumer\_Sales"

AS (SELECT "FACT\_CNSMR\_SLS\_STD"."FISC\_MO\_CD""FISC\_MO\_CD",

"FACT\_CNSMR\_SLS\_STD"."DISTR\_BE\_ID""DISTR\_BE\_ID",

"DIM\_PROD\_MV12"."FG\_CD""FG\_CD",

"DIM\_PROD\_MV12"."FG\_CD""KIT\_CD",

"DIM\_CUST\_MV13"."CUST\_CD""CUST\_CD",

"DIM\_CUST\_MV13"."CUST\_NM""CUST\_NM",

"DIM\_CUST\_MV13"."CUST\_ADDR\_LN1\_TXT""CUST\_ADDR\_LN1\_TXT",

"DIM\_CUST\_MV13"."CUST\_CITY\_NM""CUST\_CITY\_NM",

"DIM\_CUST\_MV13"."CUST\_STT\_CD""CUST\_STT\_CD",

"DIM\_CUST\_MV13"."CUST\_ZIP\_CD""CUST\_ZIP\_CD",

"DIM\_CUST\_MV13"."COT\_CD""COT\_CD",

"DIM\_CUST\_MV13"."COT\_NM""COT\_NM",

"FACT\_CNSMR\_SLS\_STD"."SS\_QTY""SS\_QTY"

FROM "DIM\_PROD\_MV12"

LEFT OUTER JOIN

( "DIM\_CUST\_MV13"

INNER JOIN

"TRANSDATA"."FACT\_CNSMR\_SLS\_STD""FACT\_CNSMR\_SLS\_STD"

ON "DIM\_CUST\_MV13"."OBJECT\_ID" =

"FACT\_CNSMR\_SLS\_STD"."CUST\_BE\_ID")

ON "DIM\_PROD\_MV12"."BE\_ID" =

"FACT\_CNSMR\_SLS\_STD"."KIT\_BE\_ID")

SELECT DISTINCT SUBSTR ("EASI\_Sales"."YR\_MTH\_CD", 1, 6)

"E\_YR\_MTH\_CD",

"EASI\_Sales"."CUST\_ACCT\_BE\_ID""E\_CUST\_ACCT\_BE\_ID",

"EASI\_Sales"."FG\_CD""E\_FG\_CD",

"EASI\_Sales"."CUST\_NM""E\_CUST\_NM",

"Consumer\_Sales"."FISC\_MO\_CD""C\_FISC\_MO\_CD",

"Consumer\_Sales"."DISTR\_BE\_ID""C\_DISTR\_BE\_ID",

"Consumer\_Sales"."FG\_CD""C\_FG\_CD",

"Consumer\_Sales"."KIT\_CD""KIT\_CD",

"Consumer\_Sales"."CUST\_NM""C\_CUST\_NM",

"Consumer\_Sales"."CUST\_ADDR\_LN1\_TXT""CUST\_ADDR\_LN1\_TXT",

"Consumer\_Sales"."CUST\_CITY\_NM""CUST\_CITY\_NM",

"Consumer\_Sales"."CUST\_STT\_CD""CUST\_STT\_CD",

"Consumer\_Sales"."CUST\_ZIP\_CD""CUST\_ZIP\_CD",

"Consumer\_Sales"."COT\_CD""COT\_CD",

"Consumer\_Sales"."COT\_NM""COT\_NM",

"EASI\_Sales"."FY\_YR\_MO\_CD""E\_FY\_YR\_MO\_CD",

'DIST. RPTD'"GROUP17",

"Consumer\_Sales"."CUST\_CD""C\_CUST\_CD",

"EASI\_Sales"."CUST\_CD""E\_CUST\_CD",

"Consumer\_Sales"."SS\_QTY""SS\_QTY"

FROM "EASI\_Sales", "Consumer\_Sales"

WHERE "EASI\_Sales"."FY\_YR\_MO\_CD">= SUBSTR ('01012011', 1, 6)

AND "EASI\_Sales"."YR\_MTH\_CD"<= "Consumer\_Sales"."FISC\_MO\_CD"

AND ("EASI\_Sales"."FG\_CD" = "Consumer\_Sales"."FG\_CD"

OR "EASI\_Sales"."FG\_CD" = "Consumer\_Sales"."KIT\_CD")

AND "EASI\_Sales"."CUST\_ACCT\_BE\_ID" = "Consumer\_Sales"."DISTR\_BE\_ID"

RATIONALE: SQL statement with SQL\_ID "4p0xrmd31t34c" was executed 1

times and had an average elapsed time of 15838 seconds.

RATIONALE: Waiting for event "PX Deq Credit: send blkd" in wait class

"Other" accounted for 65% of the database time spent in processing

the SQL statement with SQL\_ID "4p0xrmd31t34c".

Detail for Selected 5 Minute Interval

Start TimeNov 6, 2014 8:29:35 AM GMT

Top SQL

Actions [Go]

Select All | Select None

Select Activity (%) [Sorted in descending order] SQL ID SQL Type

[CPU (10.24%)] [Concurrency (4.01%)] [Other (.04%)] 14.29 57w71dgk5qbtx SELECT

[Concurrency (14.04%)] [Other (.06%)] [CPU (.00%)] 14.11 459f3z9u4fb3u SELECT

[Concurrency (3.65%)] [Other (.05%)] [CPU (.02%)] 3.72 3cj240n6v9znj PL/SQL EXECUTE

[Network (1.97%)] [CPU (.04%)] [User I/O (.00%)] 2.02 3h6gj58wp9u1y PL/SQL EXECUTE

[User I/O (1.02%)] [Concurrency (.42%)] [Other (.05%)] [CPU (.02%)] 1.51 bzhtzu2yf68jb SELECT

select \* from table(dbms\_xplan.display\_cursor('&sql\_id',null,'AdVanced ALLSTATS LAST'));

select \* from table(dbms\_xplan.display\_awr('&sql\_id',null,null,'advanced'));

3 - filter(("KGLHDNSP"=7 AND "KGLNAOBJ" LIKE 'ORA$ALERT$%' AND BITAND("KGLHDFLG",128)<>0))

Column Projection Information (identified by operation id):

-----------------------------------------------------------

1 - (#keys=1) STRDEF[990]

2 - STRDEF[990]

3 - "KGLNAOBJ"[VARCHAR2,1000], "KGLHDNSP"[NUMBER,22], "KGLHDFLG"[NUMBER,22]

4 - "SID"[VARCHAR2,30]

Note

-----

- cardinality feedback used for this statement

- Warning: basic plan statistics not available. These are only collected when:

\* hint 'gather\_plan\_statistics' is used for the statement or

\* parameter 'statistics\_level' is set to 'ALL', at session or system level

SQL\_ID 57w71dgk5qbtx, child number 7

-------------------------------------

SELECT DISTINCT SUBSTR(KGLNAOBJ,11) SID FROM X$KGLOB WHERE KGLHDNSP = 7

AND KGLNAOBJ LIKE 'ORA$ALERT$%' AND BITAND(KGLHDFLG,128)!=0 UNION

SELECT DISTINCT SID FROM DBMS\_ALERT\_INFO

Plan hash value: 2001468060

---------------------------------------------------------------------------------------------------------------------

| Id | Operation | Name | E-Rows |E-Bytes| Cost (%CPU)| E-Time | OMem | 1Mem | Used-Mem |

---------------------------------------------------------------------------------------------------------------------

| 0 | SELECT STATEMENT | | | | 9 (100)| | | | |

| 1 | SORT UNIQUE | | 183 | 2437 | 9 (45)| 00:00:01 | 43008 | 43008 |38912 (0)|

| 2 | UNION-ALL | | | | | | | | |

|\* 3 | FIXED TABLE FULL | X$KGLOB | 1 | 71 | 2 (100)| 00:00:01 | | | |

| 4 | INDEX FAST FULL SCAN| SYS\_C005421344 | 436 | 5668 | 5 (0)| 00:00:01 | | | |

---------------------------------------------------------------------------------------------------------------------

Query Block Name / Object Alias (identified by operation id):

-------------------------------------------------------------

1 - SET$1

3 - SEL$1 / X$KGLOB@SEL$1

4 - SEL$2 / DBMS\_ALERT\_INFO@SEL$2

Outline Data

-------------

/\*+

BEGIN\_OUTLINE\_DATA

IGNORE\_OPTIM\_EMBEDDED\_HINTS

OPTIMIZER\_FEATURES\_ENABLE('11.2.0.4')

DB\_VERSION('11.2.0.4')

ALL\_ROWS

OUTLINE\_LEAF(@"SEL$1")

OUTLINE\_LEAF(@"SEL$2")

OUTLINE\_LEAF(@"SET$1")

INDEX\_FFS(@"SEL$2""DBMS\_ALERT\_INFO"@"SEL$2" ("DBMS\_ALERT\_INFO"."NAME""DBMS\_ALERT\_INFO"."SID"))

FULL(@"SEL$1""X$KGLOB"@"SEL$1")

END\_OUTLINE\_DATA

\*/

Predicate Information (identified by operation id):

---------------------------------------------------

3 - filter(("KGLHDNSP"=7 AND "KGLNAOBJ" LIKE 'ORA$ALERT$%' AND BITAND("KGLHDFLG",128)<>0))

Column Projection Information (identified by operation id):

-----------------------------------------------------------

1 - (#keys=1) STRDEF[990]

2 - STRDEF[990]

3 - "KGLNAOBJ"[VARCHAR2,1000], "KGLHDNSP"[NUMBER,22], "KGLHDFLG"[NUMBER,22]

4 - "SID"[VARCHAR2,30]

Note

-----

- cardinality feedback used for this statement

- Warning: basic plan statistics not available. These are only collected when:

\* hint 'gather\_plan\_statistics' is used for the statement or

\* parameter 'statistics\_level' is set to 'ALL', at session or system level

SQL\_ID 57w71dgk5qbtx, child number 8

-------------------------------------

SELECT DISTINCT SUBSTR(KGLNAOBJ,11) SID FROM X$KGLOB WHERE KGLHDNSP = 7

AND KGLNAOBJ LIKE 'ORA$ALERT$%' AND BITAND(KGLHDFLG,128)!=0 UNION

SELECT DISTINCT SID FROM DBMS\_ALERT\_INFO

Plan hash value: 2001468060

---------------------------------------------------------------------------------------------------------------------

| Id | Operation | Name | E-Rows |E-Bytes| Cost (%CPU)| E-Time | OMem | 1Mem | Used-Mem |

---------------------------------------------------------------------------------------------------------------------

| 0 | SELECT STATEMENT | | | | 9 (100)| | | | |

| 1 | SORT UNIQUE | | 183 | 2437 | 9 (45)| 00:00:01 | 43008 | 43008 |38912 (0)|

| 2 | UNION-ALL | | | | | | | | |

|\* 3 | FIXED TABLE FULL | X$KGLOB | 1 | 71 | 2 (100)| 00:00:01 | | | |

| 4 | INDEX FAST FULL SCAN| SYS\_C005421344 | 436 | 5668 | 5 (0)| 00:00:01 | | | |

---------------------------------------------------------------------------------------------------------------------

Query Block Name / Object Alias (identified by operation id):

-------------------------------------------------------------

1 - SET$1

3 - SEL$1 / X$KGLOB@SEL$1

4 - SEL$2 / DBMS\_ALERT\_INFO@SEL$2

Outline Data

-------------

/\*+

BEGIN\_OUTLINE\_DATA

IGNORE\_OPTIM\_EMBEDDED\_HINTS

OPTIMIZER\_FEATURES\_ENABLE('11.2.0.4')

DB\_VERSION('11.2.0.4')

ALL\_ROWS

OUTLINE\_LEAF(@"SEL$1")

OUTLINE\_LEAF(@"SEL$2")

OUTLINE\_LEAF(@"SET$1")

INDEX\_FFS(@"SEL$2""DBMS\_ALERT\_INFO"@"SEL$2" ("DBMS\_ALERT\_INFO"."NAME""DBMS\_ALERT\_INFO"."SID"))

FULL(@"SEL$1""X$KGLOB"@"SEL$1")

END\_OUTLINE\_DATA

\*/

Predicate Information (identified by operation id):

### 7) Various option of sql tracing & tkprof

There are several tracing methods available, Oracle now recommends that you use the

DBMS\_MONITOR package for most types of tracing. TKPROF is a utility that lets you format any extended trace files that you generate with the event 10046 or

through the DBMS\_MONITOR package.

In an Oracle 11.1or higher release, you can use the enhanced SQL tracing interface to trace one or more

SQL statements. Here are the steps to tracing a set of SQL statements.

1.

Issue the alter session set events statement, as shown here, to set up the

trace.

SQL> alter session set events 'sql\_trace level 12';

Session altered.

SQL>

2.

Execute the SQL statements.

SQL> select count(\*) from sales;

3.

Set tracing off.

SQL> alter session set events 'sql\_trace off';

Session altered.

SQL>

Normal users can use the DBMS\_SESSION package to trace their sessions, as shown in this example:

SQL>execute dbms\_session.session\_trace\_enable(waits=>true, binds=> false);

To disable tracing, the user must execute the session\_trace\_disable procedure, as shown here:

SQL> execute dbms\_session.session\_trace\_disable();

-- In SQL\*Plus, obtain Data Pump process info:

CONNECT / as sysdba

set lines 150 pages 100 numwidth 7

col program for a38

col username for a10

col spid for a7

select to\_char(sysdate,'YYYY-MM-DD HH24:MI:SS') "DATE", s.program, s.sid,

s.status, s.username, d.job\_name, p.spid, s.serial#, p.pid

from v$session s, v$process p, dba\_datapump\_sessions d

where p.addr=s.paddr and s.saddr=d.saddr;

SQL> conn / as sysdba

SQL> select s.sid,p.spid "ospid" , p.pid "orapid"

from v$process p, v$session s

where p.addr = s.paddr

and s.sid in (<sid\_for\_worker\_process>,<sid\_for\_master\_process>);

-- Get SQL\_TRACE Worker process with level 8:

oradebug setospid <ospid\_for\_worker>

oradebug unlimit

oradebug event 10046 trace name context forever, level 8

oradebug tracefile\_name --> trace file to be uploaded

-- Get SQL\_TRACE Master process with level 8:

oradebug setospid <ospid\_for\_master>

oradebug unlimit

oradebug event 10046 trace name context forever, level 8

oradebug tracefile\_name --> trace file to be uploaded

-- To stop the tracing:

oradebug event 10046 trace name context off

2)Get the tkprof formatted output

tkprof trcfile outfile waits=y sort=exeela

exec dbms\_support.start\_trace\_in\_session (4361,2072,binds=>true,waits=>true);

exec dbms\_support.stop\_trace\_in\_session (4361,2072);

exec dbms\_system.set\_sql\_trace\_in\_session(1279,62705,true);

EXEC DBMS\_SYSTEM.set\_sql\_trace\_in\_session(sid=>123, serial#=>1234, sql\_trace=>FALSE);

You want to examine a raw SQL trace file.

Solution

Open the trace file in a text editor to inspect the tracing information. Here are portions of a raw SQL

trace generated by executing the dbms\_monitor.session\_trace\_enable procedure:

PARSING IN CURSOR #3 len=490 dep=1 uid=85 oct=3 lid=85 tim=269523043683 hv=672110367

ad='7ff18986250' sqlid='bqasjasn0z5sz'

PARSE #3:c=0,e=647,p=0,cr=0,cu=0,mis=1,r=0,dep=1,og=1,plh=0,tim=269523043680

EXEC #3:c=0,e=1749,p=0,cr=0,cu=0,mis=1,r=0,dep=1,og=1,plh=3969568374,tim=269523045613

WAIT #3: nam='Disk file operations I/O' ela= 15833 FileOperation=2 fileno=4 filetype=2 obj#=-1

tim=269523061555

FETCH #3:c=0,e=19196,p=0,cr=46,cu=0,mis=0,r=1,dep=1,og=1,plh=3969568374,tim=269523064866

STAT #3 id=3 cnt=12 pid=2 pos=1 obj=0 op='HASH GROUP BY (cr=46 pr=0 pw=0 time=11 us cost=4

size=5317 card=409)'

STAT #3 id=4 cnt=3424 pid=3 pos=1 obj=89079 op='TABLE ACCESS FULL DEPT (cr=16 pr=0 pw=0

time=246 us cost=3 size=4251 card=327)'

As you can see from this excerpt of the raw trace file, you can glean useful information, such as

parse misses, waits, an

Formatting Trace Files with TKPROF

tkprof user\_sql\_001.trc user1.prf explain=hr/hr table=hr.temp\_plan\_table\_a sys=no

sort=exeela,prsela,fchela

In the example shown here, the tkprof command takes the user\_sql\_001.trc trace file as input and

generates an output file named user1.prf. The “How it Works” section of this recipe explains key

optional arguments of the TKPROF utility.

**tkprof rcc1\_ora\_3695308.trc rcc1\_ora\_3695308.prf SYS=NO SORT= EXECPU,FCHCPU**

or

**tkprof rcc2\_ora\_1294546.trc thirdnewelapsechela.txt waits=yes sys=no sort=exeela,fchela explain='system/'**

TKPROF: Release 10.2.0.5.0 - Production on Tue May 10 11:40:37 2016

Copyright (c) 1982, 2007, Oracle. All rights reserved.

Trace file: dmsprd\_ora\_58851386.trc

Sort options: execpu fchcpu

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

count = number of times OCI procedure was executed

cpu = cpu time in seconds executing

elapsed = elapsed time in seconds executing

disk = number of physical reads of buffers from disk

query = number of buffers gotten for consistent read

current = number of buffers gotten in current mode (usually for update)

rows = number of rows processed by the fetch or execute call

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

COMMIT

call count cpu elapsed disk query current rows

------- ------ -------- ---------- ---------- ---------- ---------- ----------

Parse 0 0.00 0.00 0 0 0 0

Execute 3994 0.77 9.98 0 0 3994 0

Fetch 0 0.00 0.00 0 0 0 0

------- ------ -------- ---------- ---------- ---------- ---------- ----------

total 3994 0.77 9.98 0 0 3994 0

Misses in library cache during parse: 0

**Tracing Parallel Queries in a RAC System**

Finding the trace files for the server (or thread or slave) processes is sometimes difficult in a RAC

environment, because you aren’t sure on which node or node(s) the database has created the trace files.

Here are the steps to follow to make it easier to find the trace files on the different nodes.

1. Set the px\_trace with an alter session command, to help identify the trace

files, as shown here:

SQL> alter session set tracefile\_identifier='10046';

SQL> alter session set "\_px\_trace" = low , messaging;

SQL> alter session set events '10046 trace name context forever,level 12';

2. Execute your parallel query.

SQL> alter table bigsales (parallel 4);

SQL> select count(\*) from bigsales;

3. Turn all tracing off.

SQL> alter session set events '10046 trace name context off';

SQL> alter session set "\_px\_trace" = none;

Specifying px\_trace will cause the query coordinator’s trace file to include information about the

slave processes that are part of the query, and the instance each slave process belongs to. You can then

retrieve the trace files from the instances listed in the query coordinator’s trace file.

**Tracing Multiple Sessions**

You can trace multiple sessions that belong to a user by using the client\_id\_trace\_enable procedure

from the DBMS\_MONITOR package. Before you can execute the dbms\_monitor.client\_id\_trace\_enable

procedure, you must set the client\_identifier for the session by using the DBMS\_SESSION package, as

shown here:

SQL> execute dbms\_session.set\_identifier('SH')

Once you set the client identifier as shown here, the client\_identifier column in the V$SESSION

view is populated. You can confirm the value of the client\_identifier column by executing the

following statement:

SQL> select sid, serial#,username from v$session where client\_identifier='SH';

Now you can execute the dbms\_monitor.client\_id\_trace\_enable procedure:

SQL> execute dbms\_monitor.client\_id\_trace\_enable(client\_id=>'SH', waits=>true, binds=>false);

You can disable the trace with the following command:

SQL> execute dbms\_monitor.client\_id\_trace\_disable(client\_id=>'SH');

[oracle@ibmlab trace]$

**8) Various option for explain plan generation:**

The EXPLAINPLAN statement displays execution plans chosen by the Oracle optimizer for SELECT, UPDATE, INSERT, and DELETE statements. A statement's execution plan is the sequence of operations Oracle performs to run the statement.

The row source tree is the core of the execution plan. It shows the following information:

* An ordering of the tables referenced by the statement
* An access method for each table mentioned in the statement
* A join method for tables affected by join operations in the statement
* Data operations like filter, sort, or aggregation

In addition to the row source tree, the plan table contains information about the following:

* Optimization, such as the cost and cardinality of each operation
* Partitioning, such as the set of accessed partitions
* Parallel execution, such as the distribution method of join inputs

The EXPLAINPLAN results let you determine whether the optimizer selects a particular execution plan, such as, nested loops join. It also helps you to understand the optimizer decisions, such as why the optimizer chose a nested loops join instead of a hash join, and lets you understand the performance of a query.

1. EXPLAIN PLAN command

2. V$SQL\_PLAN

3. Automatic Workload Repository (AWR)

4. SQL Tuning Set (STS)

5. SQL Plan Baseline (SPM)

SQL> set autotrace traceonly explain

SQL> select \* from emp;

Execution Plan

----------------------------------------------------------

Plan hash value: 3956160932

--------------------------------------------------------------------------

| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time |

--------------------------------------------------------------------------

| 0 | SELECT STATEMENT | | 14 | 1218 | 3 (0)| 00:00:01 |

| 1 | TABLE ACCESS FULL| EMP | 14 | 1218 | 3 (0)| 00:00:01 |

--------------------------------------------------------------------------

Note

-----

- dynamic statistics used: dynamic sampling (level=2)

SQL> select \* from dept;

Execution Plan

----------------------------------------------------------

Plan hash value: 3383998547

--------------------------------------------------------------------------

| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time |

--------------------------------------------------------------------------

| 0 | SELECT STATEMENT | | 4 | 120 | 3 (0)| 00:00:01 |

| 1 | TABLE ACCESS FULL| DEPT | 4 | 120 | 3 (0)| 00:00:01 |

--------------------------------------------------------------------------

Note

-----

- dynamic statistics used: dynamic sampling (level=2)

SQL> select count(\*) from emp;

Execution Plan

----------------------------------------------------------

Plan hash value: 2083865914

-------------------------------------------------------------------

| Id | Operation | Name | Rows | Cost (%CPU)| Time |

-------------------------------------------------------------------

| 0 | SELECT STATEMENT | | 1 | 3 (0)| 00:00:01 |

| 1 | SORT AGGREGATE | | 1 | | |

| 2 | TABLE ACCESS FULL| EMP | 14 | 3 (0)| 00:00:01 |

-------------------------------------------------------------------

Note

-----

- dynamic statistics used: dynamic sampling (level=2)

SQL> select ename from emp;

Execution Plan

----------------------------------------------------------

Plan hash value: 3956160932

--------------------------------------------------------------------------

| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time |

--------------------------------------------------------------------------

| 0 | SELECT STATEMENT | | 14 | 98 | 3 (0)| 00:00:01 |

| 1 | TABLE ACCESS FULL| EMP | 14 | 98 | 3 (0)| 00:00:01 |

--------------------------------------------------------------------------

Note

-----

- dynamic statistics used: dynamic sampling (level=2)

SQL>

**9) Test case of your analysis of SQLT, AWR report, explain plan analysis:**

SQLT, also known as SQLTXPLAIN is a tool provided by Oracle Server Technologies Center of Expertise that can be used to diagnose why a particular SQL statement is performing poorly. It is not like AWR or Statspack which provide a system-wide view of performance. SQLT is very focused and works on one SQL statement at a time. SQLT inputs one SQL statement and outputs a set of diagnostics files. These files can be used to diagnose SQL statements performing poorly.

starting with Oracle 10.2 there is a script ORACLE\_HOME/rdbms/admin/sqltrpt.sql which can be used for usage of SQL Tuning Advisor from the command line and it will give recommendation for **problematic** sql statements.

[oracle@ibmlab ~]$ sqlplus / as sysdba

SQL\*Plus: Release 12.1.0.2.0 Production on Wed Jun 15 16:28:45 2016

Copyright (c) 1982, 2014, Oracle. All rights reserved.

Connected to:

Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production

With the Partitioning, OLAP, Advanced Analytics and Real Application Testing options

SQL> @?/rdbms/admin/sqltrpt.sql

15 Most expensive SQL in the cursor cache

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

SQL\_ID ELAPSED SQL\_TEXT\_FRAGMENT

------------- ---------- -------------------------------------------------------

5yv7yvjgjxugg 196.57 select TIME\_WAITED\_MICRO from V$SYSTEM\_EVENT where eve

22356bkgsdcnh 25.23 SELECT COUNT(\*) FROM X$KSPPI A, X$KSPPCV2 B WHERE A.IND

c9umxngkc3byq 19.07 select sql\_id, sql\_exec\_id, dbop\_name, dbop\_exec\_id, to

1p5grz1gs7fjq 17.95 select obj#,type#,ctime,mtime,stime, status, dataobj#,

32qq8k1n8ynn9 16.27 Select BYTES, extents from dba\_segments where OWNER =

04kug40zbu4dm 14.33 select policy#, action# from aud\_object\_opt$ where obje

772s25v1y0x8k 14.00 select shared\_pool\_size\_for\_estimate s, shared

aykvshm7zsabd 10.99 select size\_for\_estimate, size\_fac

fhf8upax5cxsz 10.78 BEGIN sys.dbms\_auto\_report\_internal.i\_save\_report (:rep

49s332uhbnsma 10.71 declare vsn varchar2(20); b

1fvsn5j51ugz3 10.28 begin dbms\_rcvman.resetAll; end;

SQL\_ID ELAPSED SQL\_TEXT\_FRAGMENT

------------- ---------- -------------------------------------------------------

g92kfgtvgpakv 10.19 select i.obj#,i.ts#,i.file#,i.block#,i.intcols,i.type#,

0w26sk6t6gq98 7.08 SELECT XMLTYPE(DBMS\_REPORT.GET\_REPORT\_WITH\_SUMMARY(:B1

28bgqbzpa87xf 6.74 declare policy varchar2(512);

dfffkcnqfystw 6.52 WITH MONITOR\_DATA AS (SELECT INST\_ID, KEY, NVL2(PX\_QCSI

15 Most expensive SQL in the workload repository

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

ERROR:

ORA-01427: single-row subquery returns more than one row

Specify the Sql id

~~~~~~~~~~~~~~~~~~

Specify the Sql id

~~~~~~~~~~~~~~~~~~

Enter value for sqlid: 5yv7yvjgjxugg

Sql Id specified: 5yv7yvjgjxugg

Tune the sql

~~~~~~~~~~~~

GENERAL INFORMATION SECTION

-------------------------------------------------------------------------------

Tuning Task Name : TASK\_178

Tuning Task Owner : SYS

Workload Type : Single SQL Statement

Scope : COMPREHENSIVE

Time Limit(seconds): 1800

Completion Status : COMPLETED

Started at : 06/15/2016 16:38:42

Completed at : 06/15/2016 16:38:43

-------------------------------------------------------------------------------

Schema Name : SYS

Container Name: CDB$ROOT

SQL ID : 5yv7yvjgjxugg

SQL Text : select TIME\_WAITED\_MICRO from V$SYSTEM\_EVENT where event =

'Shared IO Pool Memory'

-------------------------------------------------------------------------------

There are no recommendations to improve the statement.

-------------------------------------------------------------------------------

SQL>

Explain plan analysis:

SQL> @?/rdbms/admin/sqltrpt.sql

15 Most expensive SQL in the cursor cache

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

SQL\_ID ELAPSED SQL\_TEXT\_FRAGMENT

------------- ---------- -------------------------------------------------------

7x2gtaa25snk6 448.84 update emp set sal=1000 where deptno=10

5yv7yvjgjxugg 203.77 select TIME\_WAITED\_MICRO from V$SYSTEM\_EVENT where eve

am70ur9qwgp3z 21.03 select l1.sid, ' IS BLOCKING ', l2.sid from v$lock l1

772s25v1y0x8k 14.53 select shared\_pool\_size\_for\_estimate s, shared

04kug40zbu4dm 14.41 select policy#, action# from aud\_object\_opt$ where obje

aykvshm7zsabd 11.39 select size\_for\_estimate, size\_fac

fhf8upax5cxsz 7.85 BEGIN sys.dbms\_auto\_report\_internal.i\_save\_report (:rep

8p447s6p0rv6b 4.22 select java\_pool\_size\_for\_estimate s, java\_po

0w26sk6t6gq98 3.81 SELECT XMLTYPE(DBMS\_REPORT.GET\_REPORT\_WITH\_SUMMARY(:B1

5k5207588w9ry 3.50 SELECT DBMS\_REPORT.GET\_REPORT(:B1 ) FROM DUAL

dfffkcnqfystw 3.37 WITH MONITOR\_DATA AS (SELECT INST\_ID, KEY, NVL2(PX\_QCSI

SQL\_ID ELAPSED SQL\_TEXT\_FRAGMENT

------------- ---------- -------------------------------------------------------

gd28w82ct6rva 3.11 select audit$ from tab$ where obj# = :1

6ajkhukk78nsr 1.33 begin prvt\_hdm.auto\_execute( :dbid, :inst\_num , :end\_sn

cgtc5gb7c4g07 0.94 select dbid, status\_flag from wrm$\_wr\_control order by

grjtn5w5y8jfj 0.78 SELECT XMLELEMENT( "spot\_addm", XMLPARSE(DOCUMENT DBMS\_

15 Most expensive SQL in the workload repository

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

ERROR:

ORA-01427: single-row subquery returns more than one row

Specify the Sql id

~~~~~~~~~~~~~~~~~~

Enter value for sqlid: 7x2gtaa25snk6

Sql Id specified: 7x2gtaa25snk6

Tune the sql

~~~~~~~~~~~~

GENERAL INFORMATION SECTION

------------------------------------------------------------------------------

Tuning Task Name : TASK\_192

Tuning Task Owner : SYS

Workload Type : Single SQL Statement

Scope : COMPREHENSIVE

Time Limit(seconds): 1800

Completion Status : COMPLETED

Started at : 06/15/2016 20:46:28

Completed at : 06/15/2016 20:46:29

-------------------------------------------------------------------------------

Schema Name : SCOTT

Container Name: PDB1

SQL ID : 7x2gtaa25snk6

SQL Text : update emp set sal=1000 where deptno=10

-------------------------------------------------------------------------------

FINDINGS SECTION (1 finding)

-------------------------------------------------------------------------------

1- Statistics Finding

---------------------

Table "SCOTT"."EMP" was not analyzed.

Recommendation

--------------

- Consider collecting optimizer statistics for this table.

execute dbms\_stats.gather\_table\_stats(ownname =>'SCOTT', tabname =>

'EMP', estimate\_percent => DBMS\_STATS.AUTO\_SAMPLE\_SIZE,

method\_opt =>'FOR ALL COLUMNS SIZE AUTO');

Rationale

---------

The optimizer requires up-to-date statistics for the table in order to

select a good execution plan.

-------------------------------------------------------------------------------

EXPLAIN PLANS SECTION

-------------------------------------------------------------------------------

1- Original

-----------

Plan hash value: 1494045816

---------------------------------------------------------------------------

| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time |

---------------------------------------------------------------------------

| 0 | UPDATE STATEMENT | | 3 | 78 | 3 (0)| 00:00:01 |

| 1 | UPDATE | EMP | | | | |

|\* 2 | TABLE ACCESS FULL| EMP | 3 | 78 | 3 (0)| 00:00:01 |

---------------------------------------------------------------------------

Predicate Information (identified by operation id):

---------------------------------------------------

2 - filter("DEPTNO"=10)

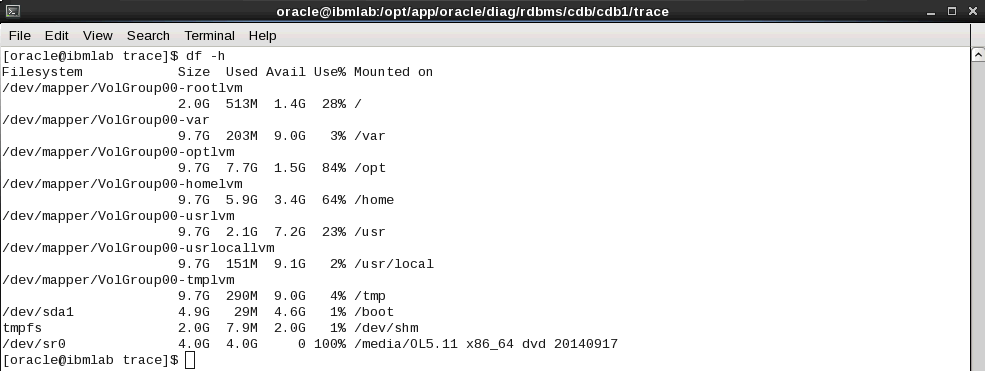
-------------------------------------------------------------------------------

SQL>

**Various OS Commands For Identifying Bottleneck**

**df -h**

We may ran out of space of the file system and the database will not even allow to login as it's not having space to generate the os level files. We can use the below commands to investigate and resolve the issue.



On the above screen shot we can clearly observed that the /opt is above 80% which is warning and we should check and remove the files which are no longer required.

**VMSTAT(virtual memory statistics)** to determine where the system is taking more resources.

/home/oracle $vmstat 5 5

System configuration: lcpu=32 mem=90112MB ent=2.00

kthr memory page faults cpu

----- ----------- ------------------------ ------------ -----------------------

r b avm fre re pi po fr sr cy in sy cs us sy id wa pc ec

5 0 20454530 211691 0 0 0 0 0 0 104 26418 1889 8 6 87 0 0.42 20.9

4 0 20454530 211691 0 0 0 0 0 0 1856 11360 3806 15 12 73 0 0.86 43.1

4 0 20454531 211687 0 0 0 0 0 0 418 21436 2441 11 7 82 0 0.58 29.0

4 0 20454531 211664 0 0 0 0 0 0 108 13954 1947 9 4 87 0 0.43 21.3

2 0 20456049 210136 0 0 0 0 0 0 81 18128 1872 8 5 88 0 0.40 19.8

The r column displays the number of processes waiting for access to a processor. The b column displays the number of processes in a sleep state. These values are usually zero.

If b(processes sleeping) and r are consistently greater than 0, then you may be using more CPU than available. .

If po (memory swapped out to disk) and pi (memory swapped in from disk) are consistently greater than 0, you may have a memory bottleneck. Paging and swapping occur when

there isn’t enough physical memory to accommodate the memory

server::/home/oracle $**vmstat -Ivt 1 10**

23068672 memory pages

22379248 lruable pages

227065 free pages

4 memory pools

3764502 pinned pages

80.0 maxpin percentage

3.0 minperm percentage

90.0 maxperm percentage

14.2 numperm percentage

3197046 file pages

0.0 compressed percentage

0 compressed pages

14.2 numclient percentage

90.0 maxclient percentage

3197046 client pages

0 remote pageouts scheduled

2028 pending disk I/Os blocked with no pbuf

828417 paging space I/Os blocked with no psbuf

2228 filesystem I/Os blocked with no fsbuf

487680 client filesystem I/Os blocked with no fsbuf

2913185 external pager filesystem I/Os blocked with no fsbuf

85.2 percentage of memory used for computational pages

server::/home/oracle

Server:/home/oracle $**vmstat -Iwt 1 10**

System configuration: lcpu=32 mem=90112MB ent=2.00

kthr memory page faults cpu time

----------- --------------------- ------------------------------------ ------------------ ----------------------- --------

r b p avm fre fi fo pi po fr sr in sy cs us sy id wa pc ec hr mi se

1 0 0 20439625 226401 11 11 0 0 0 0 61 7359 2813 8 2 90 0 0.33 16.3 08:52:32

0 0 0 20439628 226399 14 14 0 0 0 0 58 4262 1759 3 2 95 0 0.18 8.9 08:52:33

4 0 0 20439628 226399 20 20 0 0 0 0 52 4509 1752 4 2 94 0 0.19 9.3 08:52:34

11 0 0 20439628 226399 17 17 0 0 0 0 68 6426 1757 4 3 93 0 0.22 11.2 08:52:35

14 0 0 20439628 226398 10 10 0 0 0 0 63 29734 4792 10 10 80 0 0.59 29.7 08:52:36

1 0 0 20439612 226412 10 10 0 0 0 0 1257 10112 2854 9 9 81 0 0.59 29.4 08:52:37

11 0 0 20439612 226412 3 3 0 0 0 0 3214 14135 3904 17 19 65 0 1.06 52.8 08:52:38

2 0 0 20443728 222294 21 1144 0 0 0 0 2581 82473 4038 21 42 37 0 2.39 119.3 08:52:39

8 0 0 20439772 226251 18 511 0 0 0 0 3400 17234 5770 20 48 32 0 1.98 99.2 08:52:40

3 0 0 20440792 225229 11 10 0 0 0 0 894 8358 2894 12 8 80 0 0.65 32.7 08:52:41

Server[/home/fug44ppt]$ **vmstat -s**

95140704086 total address trans. faults

480621222 page ins

1113764123 page outs

3569604 paging space page ins

15652846 paging space page outs

0 total reclaims

44721429833 zero filled pages faults

844490468 executable filled pages faults

1183968453 pages examined by clock

9 revolutions of the clock hand

539831928 pages freed by the clock

1164769704 backtracks

402845 free frame waits

0 extend XPT waits

42845818 pending I/O waits

1426787329 start I/Os

590348784 iodones

162725462776 cpu context switches

9902082156 device interrupts

1745487474 software interrupts

72827743316 decrementer interrupts

2615139 mpc-sent interrupts

2850052 mpc-received interrupts

763948723 phantom interrupts

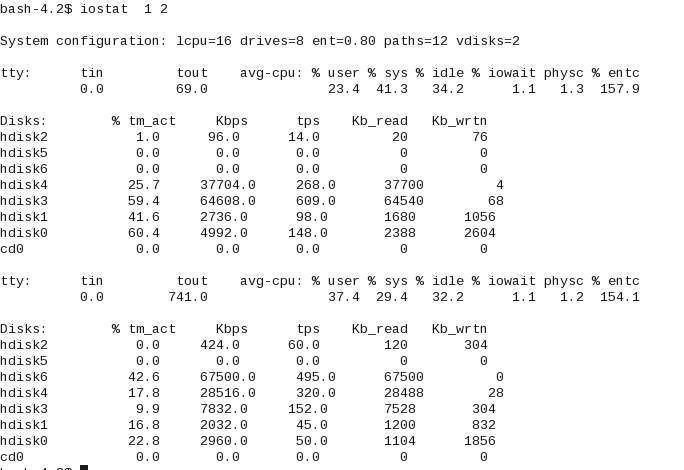
0 traps

367591277859 syscalls

Server[/home/fug44ppt]$

**I/O Stats**

Reports I/O statistics



**The prstat utility is used to identify CPU resources utilization:**

bash-2.05$ prstat 5

PID USERNAME SIZE RSS STATE PRI NICE TIME CPU PROCESS/NLWP

28195 root 71M 60M sleep 59 0 307:14:50 1.0% BESClient/4

25853 jthotaku 6624K 6464K cpu1 39 0 0:00:00 0.4% prstat/1

3796 root 232M 208M sleep 59 0 39:58:55 0.1% kuxagent/45

12 root 16M 12M sleep 59 0 3:12:36 0.1% vxconfigd/1

25832 root 5160K 3840K sleep 59 0 0:00:00 0.0% sshd/1

25846 jthotaku 2600K 2024K sleep 59 0 0:00:00 0.0% bash/1

25837 jthotaku 1936K 1512K sleep 59 0 0:00:00 0.0% ksh/1

1649 root 5296K 3288K sleep 59 0 0:00:00 0.0% automountd/2

1547 root 111M 82M sleep 59 0 0:00:50 0.0% vxsvc/18

25100 root 2400K 1968K sleep 59 0 0:00:02 0.0% inetd/1

1647 daemon 2520K 1744K sleep 59 0 0:00:00 0.0% statd/1

1507 root 3496K 1128K sleep 60 0 0:00:00 0.0% ebssdkd/2

1684 root 2152K 1320K sleep 100 - 0:00:02 0.0% xntpd/1

487 root 8904K 2832K sleep 59 0 0:00:01 0.0% vxesd/7

1527 root 13M 3152K sleep 59 0 0:00:10 0.0% pbx\_exchange/1

Total: 1828 processes, 2071 lwps, load averages: 0.04, 0.05, 0.11

**SAR:**(System Activity Reporter) SAR is a legacy system monitoring tool which is  used to report CPU activity, system loads average, memory/paging, LUN load, network activity.

bash-2.05$ sar -p 10 10

SunOS drsmgtp1 5.9 Generic\_122300-31 sun4u 06/15/2016

11:31:07 atch/s pgin/s ppgin/s pflt/s vflt/s slock/s

11:31:17 8.00 2.40 6.00 6.40 18.30 0.00

11:31:27 0.70 2.80 2.80 0.00 0.40 0.00

11:31:37 0.40 1.50 1.50 0.10 0.00 0.00

11:31:47 4.30 4.40 4.40 5.90 16.10 0.00

11:31:57 0.70 2.80 2.80 0.00 0.00 0.00

11:32:07 0.40 1.60 1.60 0.70 0.00 0.00

11:32:17 3.50 1.20 1.20 5.90 16.10 0.00

11:32:27 0.40 1.90 5.70 0.10 0.00 0.00

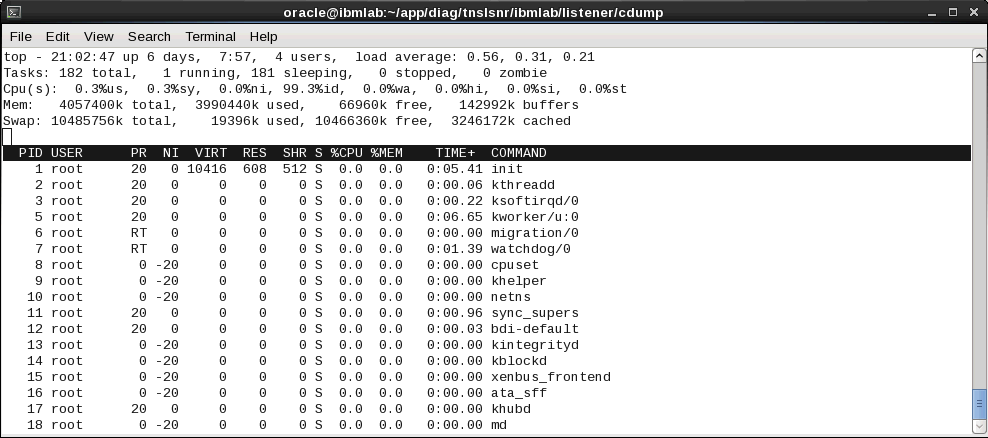
11:32:37 1.20 0.00 0.00 0.00 0.00 0.00

11:32:47 4.40 0.00 0.00 5.90 16.20 0.00

Average 2.40 1.86 2.60 2.50 6.71 0.00

bash-2.05$

**TOP** is a command to get the load information along with memory and process utilization details.



**PS:** ps is a command to quickly identify the what are the process running on the Unix an Linux servers.

[oracle@ibmlab cdump]$ ps -ef |grep pmon

oracle 23212 22307 0 21:04 pts/3 00:00:00 grep pmon

oracle 27788 1 0 Jun10 ? 00:00:20 ora\_pmon\_cdb1

[oracle@ibmlab cdump]$

**free -m:** is a command to get the memory information on the linux operating system.

[oracle@ibmlab cdump]$ free -m

total used free shared buffers cached

Mem: 3962 3897 65 0 139 3170

-/+ buffers/cache: 587 3375

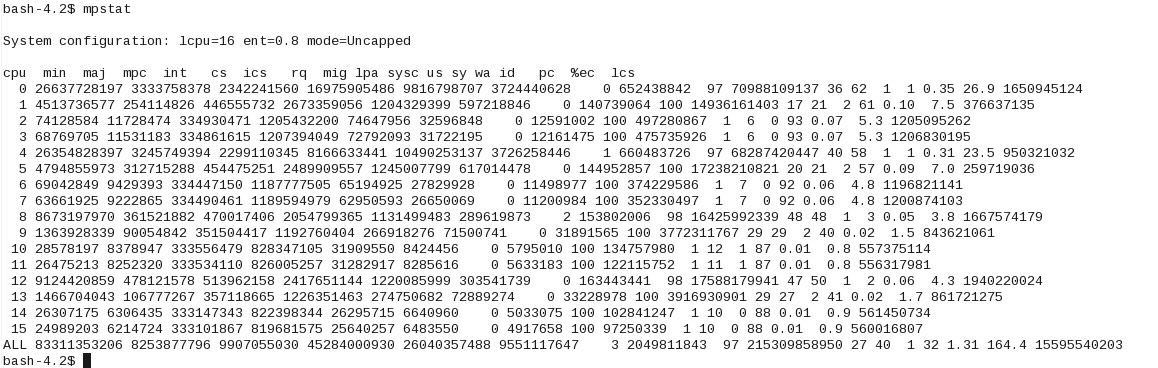
Swap: 10239 18 10221

[oracle@ibmlab cdump]$

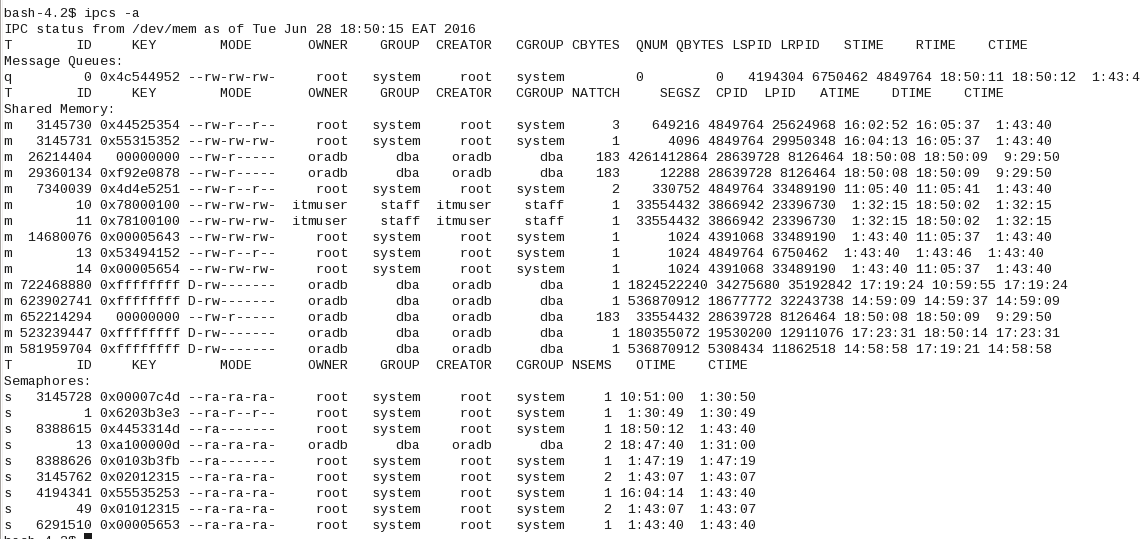
1. **The mpstat utility reports on system wide CPU statistics**

mpstat 2 10

Reports processor related statistics.

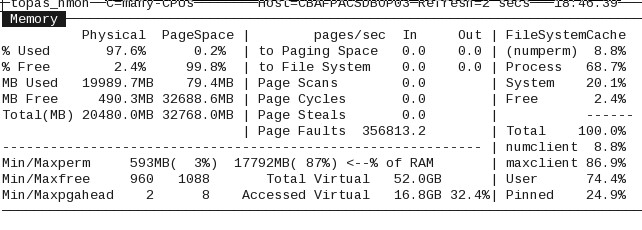


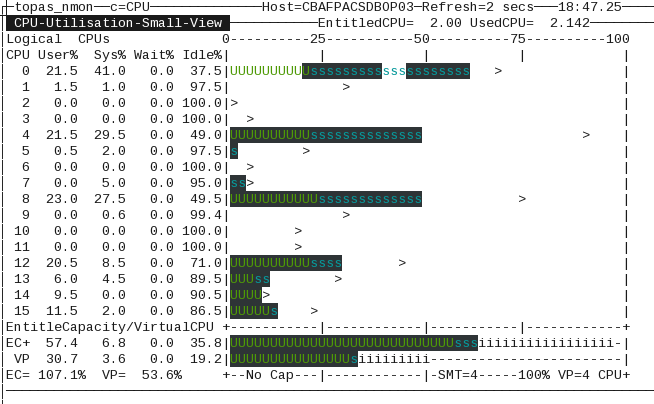
1. **Ipcs/ipcrm**

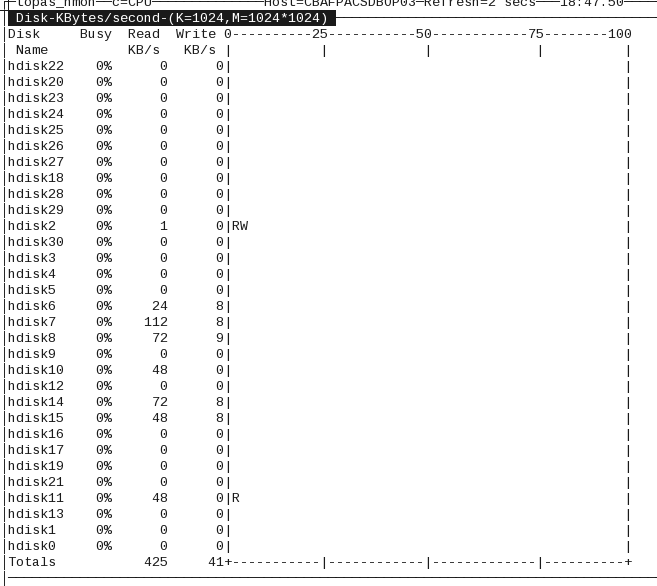


1. **NMON**

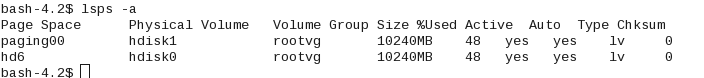
To display usages memory ,CPU ,Disk etc.



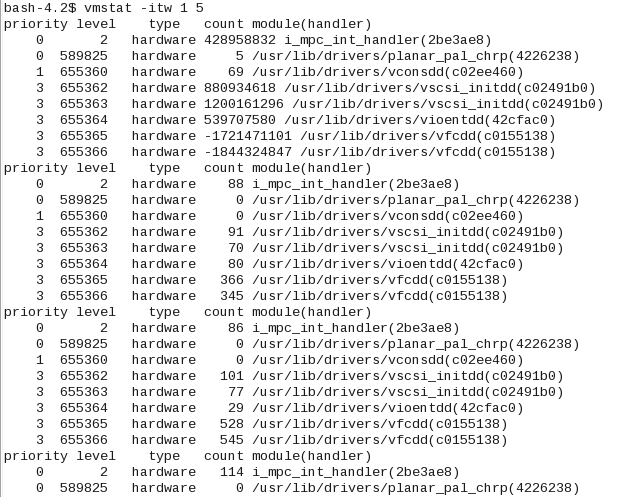




1. **Paging info**



1. **Memory and paging in one output**



**Index Rebuild Concept and Test Case**

An index is a database object that used primarily to improve the performance of SQL queries

Some of the key reasons to rebuild an index include the following:

•You want to rebuild an index that has become fragmented over time.

•You want to rebuild an index after a large, direct-path load of data.

•You want to move an index to a different tablespace.

•The index is in an unusable state because of a partition-level operation on the associated table.

The main disadvantage of the index rebuild process is that you will need space for both indexes, which is required during the rebuild process. also keep in mind that the degree of parallelism stays on the index

after the creation is complete so therefore need to change it again with no parallel or

need to reset the degree of parallelism to its original value on the index after creating it.

To ascertain index fragmentation, the following SQL statement can be used:

SQL>analyze index index\_name validate structure;

SQL>SELECT name,del\_lf\_rows,lf\_rows - del\_lf\_rows lf\_rows\_used,

to\_char(del\_lf\_rows / (lf\_rows)\*100,'999.99999') ibadness

FROM index\_stats where name ='index\_name';

if 15-20% of the table data changes, then you may consider rebuilding the index.

There are a couple of effective methods for freeing up unused space associated with an index:

• Rebuilding the index

• Shrinking the index

Before you perform either of these operations, first check USER\_SEGMENTS to verify that the amount of space used

corresponds with the Segment Advisor’s advice. In this example, the segment name is F\_REGS\_IDX1:

SQL> select bytes from user\_segments where segment\_name = 'F\_REGS\_IDX1';

BYTES

----------

166723584

This example uses the ALTER INDEX...REBUILD statement to re-organize and compact the space used by an

index:

SQL> alter index f\_regs\_idx1 rebuild;

Alternatively, use the ALTER INDEX...SHRINK SPACE statement to free up unused space in an index—for example:

SQL> alter index f\_regs\_idx1 shrink space;

Index altered.

Now query USER\_SEGMENTS again to verify that the space has been de-allocated. Here is the output for this

example:

BYTES

----------

524288

The space consumed by the index has considerably decreased.

If you use the ALTER INDEX...SHRINK SPACE operation to free up unused index space, keep in mind that

this feature requires that the target object must be created within a tablespace with automatic segment space management enabled

The clustering factor reflects how sorted the table data is with respect to the given index key. Rebuilding an index never has an influence on the clustering factor but instead requires a table re-organization.

Secondly the impact of rebuilding the index can be quite significant, please read the following comments thoroughly:

1. Most scripts around depend on the index\_stats dynamic table. This is populated by the command:

analyze index ... validate structure;

While this is a valid method to inspect the index, it grabs an exclusive table lock while analyzing the index. Especially for large indexes, this can be very dramatic, as DML operations on the table are not permitted during that time. While it can be run online without the locking considerations, it may consume additional time.

2. Redo activity may increase and general performance might be impacted as a direct result of rebuilding an index.

Insert/update/delete causes the index to evolve over time as the index splits and grows. When the index is rebuild, it will become more tightly packed; however as DML operations continue on the table, the index splits have to be redone again until the index reaches its equilibrium. As a result, the redo activity increases and the index splits are now more likely to impact performance directly as we consume more I/O, CPU, etc to serve the index restructuring. After a certain period of time the index may again experience 'issues' and may be re-flagged for a rebuild, causing the vicious cycle to continue. Therefore, it is often better to leave the index in its natural equilibrium and/or at least prevent indexes from being rebuilt on a regular basis.

3. An index coalesce is often preferred instead of an index rebuild. It has the following advantages:

- does not require approximately 2 times the disk storage

- always online

- does not restructure the index, but combines index leaf blocks as much as possible, avoiding system overhead as explained in point 2.

Generally speaking, the need to rebuild b-tree indexes is very rare, basically because a b-tree index is largely self-managed or self-balanced.

The most common justifications given for rebuilding an index are:

- index becomes fragmented

- index grows and grows - deleted space is not re-used

- index clustering factor becomes out of sync

An index coalesce is often preferred instead of an index rebuild. It has the following advantages:

does not require approximately 2 times the disk storage

- always online

- does not restructure the index, but combines index leaf blocks as much as possible, avoiding system overhead as explained in point 2.

Note: To re-allocate an index, to another tablespace for example a rebuild is required.

- deleted entries represent 20% or more of the current entries.

- the index depth is more then 4 levels.

**Lab-1**

**Check top object with high DML**

select m.table\_owner,m.table\_name, m.timestamp, sum(m.inserts), sum(m.updates), sum(m.deletes), t.num\_rows, t.last\_analyzed from sys.dba\_tab\_modifications m, dba\_tables t

where m.table\_owner = t.owner and m.table\_name = t.table\_name and m.table\_owner not in('SYS','OUTLN','SYSTEM','TSMSYS','DBSNMP','WMSYS','EXFSYS','XDB','OLAPSYS','MDSYS','CAGAUDIT','SYSMAN','DMSYS','CTXSYS','ORDSYS')

group by m.table\_owner,m.table\_name, m.timestamp, t.num\_rows,t.last\_analyzed order by 1,2

**Check specific table now**

select m.table\_owner,m.table\_name, m.timestamp, sum(m.inserts), sum(m.updates), sum(m.deletes), t.num\_rows, t.last\_analyzed from sys.dba\_tab\_modifications m, dba\_tables t

where m.table\_owner = t.owner and m.table\_name = t.table\_name and m.table\_name in('PAYMENT\_PLAN\_CREDIT\_SHARE','ZEUS\_BUILD\_FAILURES') group by m.table\_owner, m.table\_name, m.timestamp, t.num\_rows, t.last\_analyzed order by 1,2

4 /

TABLE\_NAME TIMESTAMP SUM(M.INSERTS) SUM(M.UPDATES) SUM(M.DELETES) NUM\_ROWS LAST\_ANALYZ

------------------------------ ----------- -------------- -------------- -------------- ---------- -----------

PAYMENT\_PLAN\_CREDIT\_SHARE 23-may-2017 266574883 265166199 21923 2334869164 23-nov-2016

select bytes/1024/1024/1024 from dba\_segments where segment\_name='PAYMENT\_PLAN\_CREDIT\_SHARE';

BYTES/1024/1024/1024

--------------------

371.484375

**Check indexes on Table**

SELECT t.table\_name, i.index\_name,i.blevel,i.status,t.last\_analyzed FROM dba\_tables t, dba\_indexes i WHERE t.table\_name = i.table\_name AND t.table\_name ='PAYMENT\_PLAN\_CREDIT\_SHARE';

TABLE\_NAME INDEX\_NAME BLEVEL STATUS LAST\_ANALYZ

------------------------------ ------------------------------ ---------- -------- -----------

PAYMENT\_PLAN\_CREDIT\_SHARE PF\_PAY\_PLAN\_CRED\_S\_03 3 VALID 23-nov-2016

PAYMENT\_PLAN\_CREDIT\_SHARE PF\_PAY\_PLAN\_CRED\_S\_05 3 VALID 23-nov-2016

PAYMENT\_PLAN\_CREDIT\_SHARE PF\_PAY\_PLAN\_CRED\_S\_04 3 VALID 23-nov-2016

PAYMENT\_PLAN\_CREDIT\_SHARE PK\_PAY\_PLAN\_CRED\_S 3 VALID 23-nov-2016

PAYMENT\_PLAN\_CREDIT\_SHARE PF\_PAY\_PLAN\_CRED\_S\_01 3 VALID 23-nov-2016

PAYMENT\_PLAN\_CREDIT\_SHARE PF\_PAY\_PLAN\_CRED\_S\_02 3 VALID 23-nov-2016

select owner,segment\_name,segment\_type,bytes/1024/1024/1024 from dba\_segments where segment\_name in ('PF\_PAY\_PLAN\_CRED\_S\_03','PF\_PAY\_PLAN\_CRED\_S\_05','PF\_PAY\_PLAN\_CRED\_S\_04','PK\_PAY\_PLAN\_CRED\_S','PF\_PAY\_PLAN\_CRED\_S\_01','PF\_PAY\_PLAN\_CRED\_S\_02');

OWNER SEGMENT\_NAME SEGMENT\_TYPE BYTES/1024/1024/1024

------------------------------ --------------------------------------------------------------------------------- ------------------ --------------------

PILOT\_DBA PF\_PAY\_PLAN\_CRED\_S\_05 INDEX 136.523438

PILOT\_DBA PF\_PAY\_PLAN\_CRED\_S\_04 INDEX 125.683594

PILOT\_DBA PF\_PAY\_PLAN\_CRED\_S\_03 INDEX 166.503906

PILOT\_DBA PF\_PAY\_PLAN\_CRED\_S\_02 INDEX 114.648438

PILOT\_DBA PF\_PAY\_PLAN\_CRED\_S\_01 INDEX 143.75

PILOT\_DBA PK\_PAY\_PLAN\_CRED\_S INDEX 184.082031

**Upon identification of index issue ,rebuild index.**

**Existing index can be rebuilt quickly by using parallel option. It will use multiple processes to speed up the index rebuild process.**

spool rebuild\_index.sql

select 'ALTER INDEX '||index\_name||'.'||','|| owner||' REBUILD online nologging PARALLEL 8; ' from dba\_indexes where owner='DMS\_USER' AND INDEX\_NAME IN('PRCL\_ITEM\_PRCL\_FK\_I','CUST\_REQ\_CUSTORD\_FK\_I','CUSTORD\_PK','PRCL\_ITEM\_PK','PRCL\_ITEM\_PCDR\_DATE\_I','PRCL\_PK','CUST\_REQIT\_CUST\_REQ\_FK\_I','PRCL\_ITEM\_PP\_FK\_I',

'PRCL\_ITEM\_WMS\_STAT\_I','WMSORD\_PK','P\_LOCN\_DC\_LOCN\_FK\_I','PLS\_PROD\_I','WSF\_PROD\_I','CUST\_REQIT\_PP\_FK\_I') order by 3;

spool off;

ALTER INDEX LITTLEWOODS.WEBREP\_DEPOT\_QUERY\_UPI\_IDX REBUILD ONLINE tablespace TSPACEI02A PARALLEL 4 NOLOGGING;

ALTER INDEX LITTLEWOODS.WEBREP\_DEPOT\_QUERY\_UPI\_IDX NOPARALLEL LOGGING;

ALTER INDEX LITTLEWOODS.CO\_CN\_IX REBUILD ONLINE tablespace TSPACEI02A PARALLEL 4 NOLOGGING;

ALTER INDEX LITTLEWOODS.CO\_CN\_IX NOPARALLEL LOGGING;

ALTER INDEX LITTLEWOODS.CO\_DE\_FK\_I REBUILD ONLINE tablespace TSPACEI02A PARALLEL 4 NOLOGGING;

ALTER INDEX LITTLEWOODS.CO\_DE\_FK\_I NOPARALLEL LOGGING;

ALTER INDEX LITTLEWOODS.PK\_IPB REBUILD ONLINE tablespace TSPACEI02A PARALLEL 4 NOLOGGING;

ALTER INDEX LITTLEWOODS.PK\_IPB NOPARALLEL LOGGING;

S set linesize 200

SQL>

SQL> select owner,segment\_name,SEGMENT\_TYPE,bytes/1024/1024 from dba\_segments where tablespace\_name='TSPACEI02A';

exec dbms\_stats.gather\_index\_stats('LITTLEWOODS', 'CO\_DE\_FK\_I');

exec dbms\_stats.gather\_index\_stats('LITTLEWOODS', 'CO\_RE\_FK\_I');

exec dbms\_stats.gather\_index\_stats('LITTLEWOODS', 'CO\_CN\_IX');

exec dbms\_stats.gather\_index\_stats('LITTLEWOODS', 'CO\_CU\_FK\_I')

Release space from table space if required

COL TABLESPACE\_NAME FORMAT A30

select

a.TABLESPACE\_NAME,

a.TOTAL\_IN\_GB,

to\_char(nvl((b.used),0.00),'999990.99') USED\_IN\_GB,

to\_char(nvl((c.free),0.00),'999990.99') FREE\_IN\_GB,

to\_char(nvl(((b.used/a.TOTAL\_IN\_GB)\*100),0.00),'99990.99') PCT\_USED

from

(select TABLESPACE\_NAME,

to\_char(sum(bytes)/(1024\*1024\*1024),'9999990.99') TOTAL\_IN\_GB

from sys.dba\_data\_files

group by TABLESPACE\_NAME) a,

(select TABLESPACE\_NAME,bytes/(1024\*1024\*1024) used

from sys.SM$TS\_USED) b,

(select TABLESPACE\_NAME,bytes/(1024\*1024\*1024) free

from sys.SM$TS\_free) c

where a.TABLESPACE\_NAME=b.TABLESPACE\_NAME(+) and

a.TABLESPACE\_NAME=c.tablespace\_name(+)

TABLESPACE\_NAME TOTAL\_IN\_MB USED\_IN\_MB FREE\_IN\_MB PCT\_USED

------------------------------ ----------- ---------- ---------- ---------

TSP\_ANNOTATION\_I 60.94 18.85 41.11 30.93

TSP\_TC\_ACCOUNT\_SNAPSHOT 64.94 20.51 41.99 31.58

TSP\_PARCEL\_ITEM\_I 83.08 25.88 55.66 31.15

TSP\_ORDER\_REQUEST\_I 75.99 28.32 46.58 37.27

TSP\_LARGE 3552.47 2323.34 1618.65 65.40

TSP\_MEDIUM 1393.37 980.16 412.52 70.34

TSP\_ANNOTATION 50.60 31.25 14.65 61.76

TSP\_LARGE\_ARC 46.94 9.28 37.60 19.76

TSP\_ORDER\_ITEM\_I 235.02 51.76 179.98 22.02

TSP\_MINIMUM 7.91 7.07 3.18 89.39

TSP\_PARCEL\_ITEM 28.26 21.48 3.91 76.02

For reclaiming space from Tablespace ,We need to follow below steps ,As this is large tablespace ,

We need to move huge number of tables , indexes LOBINDEX and LOBSEGMENT whose details are below

select count(1) from segment\_type from dba\_segments group by segment\_type.

COUNT(1) SEGMENT\_TYPE

---------- ------------------

367 INDEX

22 LOBINDEX

22 LOBSEGMENT

377 TABLE

eg

1) Create New Tablespace TSP\_LARGE\_NEW with appropriate size

2) Move a table to another tablespace, issue the following command:

ALTER TABLE PAYMENT\_PLAN\_CREDIT\_SHARE MOVE TABLESPACE TSP\_LARGE\_NEW;

3) To move an index, use the following:

alter index PILOT\_DBA.PK\_PAY\_PLAN\_CRED\_S parallel 8 TSP\_LARGE\_NEW

alter index PILOT\_DBA.PF\_PAY\_PLAN\_CRED\_S\_05 parallel 8 TSP\_LARGE\_NEW

alter index PILOT\_DBA.PF\_PAY\_PLAN\_CRED\_S\_04 parallel 8 TSP\_LARGE\_NEW

alter index PILOT\_DBA.PF\_PAY\_PLAN\_CRED\_S\_03 parallel 8 TSP\_LARGE\_NEW

alter index PILOT\_DBA.PF\_PAY\_PLAN\_CRED\_S\_02 parallel 8 TSP\_LARGE\_NEW

alter index PILOT\_DBA.PF\_PAY\_PLAN\_CRED\_S\_01 parallel 8 TSP\_LARGE\_NEW

alter index PILOT\_DBA.PK\_PAY\_PLAN\_CRED\_S noparallel

alter index PILOT\_DBA.PF\_PAY\_PLAN\_CRED\_S\_05 noparallel

alter index PILOT\_DBA.PF\_PAY\_PLAN\_CRED\_S\_04 noparallel

alter index PILOT\_DBA.PF\_PAY\_PLAN\_CRED\_S\_03 noparallel

alter index PILOT\_DBA.PF\_PAY\_PLAN\_CRED\_S\_02 noparallel

alter index PILOT\_DBA.PF\_PAY\_PLAN\_CRED\_S\_01 noparallel

4) To move the LOB when moving the table, use the following:

ALTER TABLE table\_name MOVE TABLESPACE TSP\_LARGE

LOB (lob\_item) STORE AS (TABLESPACE TSP\_LARGE\_NEW);

5) Drop old Tablespace TSP\_LARGE.

DROP TABLESPACE tspacei02 including contents and datafiles;

DROP TABLESPACE tspacei02 including datafiles;

**There are several ways to reclaim wasted space in a table but From Oracle 10g onwards ,this is best method to reclaim space .**

1. Enable row movement for the table.

2.Use the ALTER TABLE...SHRINK SPACE statement to free up unused space.

Note The shrink table feature requires that the table’s tablespace use automatic space segment management.

When you shrink a table, this requires that rows (if any) be moved which requires that row movement be enabled:

SQL> alter table <table\_name> enable row movement;

Next the table shrink operation is executed via an ALTER TABLE statement:

SQL> alter table <table\_name> shrink space;

You can also shrink the space associated with any index segments via the CASCADE clause:

**SQL> alter table <table Name> shrink space cascade**

**Tkprof Interpretation**

**Tkprof is an executable that 'parses' Oracle trace files to produce more readable output.**

**Remember that all the information in TkProf is available from the base trace file.**

**TKPROF allows you to analyse a trace file to determine where time is being spent and what query plans are being used on SQL statements.**

**If you have a system that is performing badly, a good way to identify problem SQL statements is to trace a typical user session and then use TkProf to format the output using the sort functions on the tkprof command line.**

**Find the appropriate trace file (In USER\_DUMP\_DEST, default**

**$ORACLE\_HOME/rdbms/log on Unix).**

**You can find the most recent trace files on Unix with the command:**

**ls -ltr**

**This will list the most recent files LAST**

**Run tkprof on the trace file thus:**

**tkprof tracefile outfile [explain=user/password] [options...]**

**Convert the trace file into tkprof format using the command:**

**tkprof <filename.trc> <output\_filename\_SORT.txt> explain=apps/<password> sort='(prsela,exeela,fchela)'**

**There are a huge number of sort options that can be accessed by simply typing 'TkProf' at the command prompt.**

**A useful starting point is the**

**'fchela' sort option which orders the output by elapsed time fetching (rememberthat timing information is only available with timed\_statistics set to true at the database level). The resultant .prf file will contain the most time consuming SQL statement at the start of the file.**

**Another useful parameter is sys. This can be used to prevent SQL statements run as user SYS from being displayed. This can make the output file much shorter and easier to manage.**

**TKPROF Options**

**~~~~~~~~~~~~~~**

**print=integer List only the first 'integer' SQL statements.**

**insert=filename List SQL statements and data inside INSERT statements.**

**sys=no TKPROF does not list SQL statements run as user SYS.**

**record=filename Record statements found in the trace file.**

**sort=option Set of zero or more of the following sort options:**

**prscnt number of times parse was called**

**prscpu cpu time parsing**

**prsela elapsed time parsing**

**prsdsk number of disk reads during parse**

**prsqry number of buffers for consistent read during parse**

**prscu number of buffers for current read during parse**

**prsmis number of misses in library cache during parse**

**execnt number of execute was called**

**execpu cpu time spent executing**

**exeela elapsed time executing**

**exedsk number of disk reads during execute**

**exeqry number of buffers for consistent read during execute**

**execu number of buffers for current read during execute**

**exerow number of rows processed during execute**

**exemis number of library cache misses during execute**

**fchcnt number of times fetch was called**

**fchcpu cpu time spent fetching**

**fchela elapsed time fetching**

**fchdsk number of disk reads during fetch**

**fchqry number of buffers for consistent read during fetch**

**fchcu number of buffers for current read during fetch**

**fchrow number of rows fetched**

**userid userid of user that parsed the cursor**

**Again, remember to always check that the TIMED\_STATISTICS parameter is set to TRUE as otherwise no time based comparisons can be made.**

**Interpreting TkProf Output Guidelines**

**=====================================**

**Column Meanings**

**===============**

**call : Statisics for each cursor's activity are divided in to 3 areas:**

**Parse: statisitics from parsing the cursor. This includes information for plan generation etc.**

**Execute: statisitics for the exection phase of a cursor**

**Fetch : statistics for actually fetching the rows**

**count : number of times we have performed a particular activity on this particular cursor**

**cpu: cpu time used by this cursor**

**elapsed: elapsed time for this cursor**

**disk: This indicates the number of blocks read from disk. Generally you want to see blocks being read from the buffer cache rather than disk.**

**query : This column is incremented if a buffer is read in Consistent mode.**

**A Consistent mode buffer is one that has been generated to give a consistent read snapshot for a long running transaction. The buffer actually contains this status in its header.**

**current: This column is incremented if a buffer found in the buffer cache that is new enough for the current transaction and is in current mode (and it is not a CR buffer). This applies to buffers that have been**

**read in to the cache as well as buffers that already exist in the cache in current mode.**

**rows: Rows retrieved by this step**

**Explain plan**

**============**

**Firstly, we advise that the autotrace feature of SQL\*Plus be used on statements rather than using TkProf mainly because the TkProf output can be confusing with regard to whether the Rule or Cost Based optimizer has been used. Because TkProf explain plan does not show any costs or statistics, it is sometimes not possible to tell definitively which optimizer has been used.**

**That said, the following output from Tkprof explain plan is useful.**

**The Rows column next to the explain plan output shows the number of rows processed by that particular step. The information is gathered from the STAT lines for each cursor in the raw trace output.**

**Remember that if the cursor is not closed then you will not see any output. Setting SQL\_TRACE to false DOES NOT close PL/SQL child cursors.**

**Cursors are closed in SQL\*Plus immediately after execution.**

**TkProf Examples and Discussion**

**==============================**

**Examples:**

**Step 1 - Look at the totals at the end of the tkprof output**

**===========================================================**

**OVERALL TOTALS FOR ALL NON-RECURSIVE STATEMENTS**

**| call | count | cpu | elapsed | disk | query | current | rows |**

**|---------|-------|------|---------|---------|--------|---------|--------|**

**| Parse | [A] 7 | 1.87 | 4.53 | 385 |[G] 553 | 22 | 0 |**

**| Execute | [E] 7 | 0.03 | 0.11 | [P] 0 |[C] 0 | [D] 0 | [F] 0 |**

**| Fetch | [E] 6 | 1.39 | 4.21 | [P] 128 |[C] 820 | [D] 3 | [F] 20 |**

**--------------------------------------------------------------------------**

**Misses in library cache during parse: 5**

**Misses in library cache during execute: 1**

**8 user SQL statements in session.**

**12 internal SQL statements in session.**

**[B] 54 SQL statements in session.**

**3 statements EXPLAINed in this session.**

**1. Compare [A] & [B] to spot over parsing. In this case we**

**have 7 parses for 54 statements which is ok.**

**2. You can use [P], [C] & [D] to determine the hit ratio.**

**Hit Ratio is logical reads/physical reads:**

**Logical Reads = Consistent Gets + DB Block Gets**

**Logical Reads = query + current**

**Logical Reads = Sum[C] + Sum[D]**

**Logical Reads = 0+820 + 0+3**

**Logical Reads = 820 + 3**

**Logical Reads = 823**

**Hit Ratio = 1 - (Physical Reads / Logical Reads)**

**Hit Ratio = 1 - (Sum[P] / Logical Reads)**

**Hit Ratio = 1 - (128 / 823)**

**Hit Ratio = 1 - (0.16)**

**Hit Ratio = 0.84 or 84%**

**3. We want fetches to be less than the number of rows as this will mean we have done less work (array fetching).**

**To see this we can compare [E] and [F].**

**[E] = 6 = Number of Fetches**

**[F] = 20 = Number of Rows**

**So we are doing 6 fetches to retrieve 20 rows - not too bad.**

**If arrayfetching was configured then rows could be retrieved with less fetches.**

**Remember that an extra fetch will be done at the end to check that the end of fetch has been reached.**

**4. [G] Shows reads on the Dictionary cache for the statements.**

**- this should not be a problem on Oracle7.**

**In this case we have done 553 reads from the Library cache.**

**STEP 2 - Examine statements using high resource**

**===============================================**

**update ...**

**where ...**

**| call | count | cpu | elapsed | disk | query | current | rows |**

**|---------|-------|-----|---------|------|---------|---------|--------|**

**| Parse | 1 | 7 | 122 | 0 | 0 | 0 | 0 |**

**| Execute | 1 | 75 | 461 | 5 | [H] 297 | [I] 3 | [J] 1 |**

**| Fetch | 0 | 0 | 0 | 0 | 0 | 0 | 0 |**

**-----------------------------------------------------------------------**

**[H] shows that this query is visiting 297 blocks to find the rows to update**

**[I] shows that only 3 blocks are visited performing the update**

**[J] shows that only 1 row is updated.**

**297 block to update 1 rows is a lot.**

**Possibly there is an index missing?**

**STEP 3 - Look for over parsing**

**==============================**

**select ...**

**| call | count | cpu | elapsed | disk | query | current | rows |**

**|---------|-------|---------|---------|------|--------|---------|-------|**

**| Parse | [M] 2 | [N] 221 | 329 | 0 | 45 | 0 | 0 |**

**| Execute | [O] 3 | [P] 9 | 17 | 0 | 0 | 0 | 0 |**

**| Fetch | 3 | 6 | 8 | 0 | [L] 4 | 0 | [K] 1 |**

**-------------------------------------------------------------------------**

**Misses in library cache during parse: 2 [Q]**

**[K] is shows that the query has returned 1 row.**

**[L] shows that we had to read 4 blocks to get this row back.**

**This is fine.**

**[M] show that we are parsing the statement twice - this is not desirable especially as the cpu usage is high [N] in comparison to the execute**

**figures : [O] & [P]. [Q] shows that these parses are hard parses. If**

**[Q] was 1 then the statement would have had 1 hard parse followed by**

**a soft parse (which just looks up the already parsed detail in the library cache). See Note:32895.1 for more details.**

**This is not a particularly bad example since the query has only been executed a few times. However excessive parsing should be avoided as far as possible by:**

**o Ensuring that code is shared:**

**- use bind variables**

**- make shared pool large enough to hold query definitions in memory long enough to be reused.**

**Running TKPROF on a SQL Trace File**

For SELECT statements, the number of rows returned appears for the fetch step.

For UPDATE, DELETE, and INSERT statements, the number of rows processed appears for the execute step

-Note-

**Row Source Operations**

Row source operations provide the number of rows processed for each operation executed on the rows and additional row source information, such as physical reads

and writes.

The following is a sample:

Rows Row Source Operation

------- ---------------------------------------------------

0 DELETE (cr=43141 r=266947 w=25854 time=60235565 us)

28144 HASH JOIN ANTI (cr=43057 r=262332 w=25854 time=48830056 us)

51427 TABLE ACCESS FULL STATS$SQLTEXT (cr=3465 r=3463 w=0 time=865083 us)

647529 INDEX FAST FULL SCAN STATS$SQL\_SUMMARY\_PK

(cr=39592 r=39325 w=0 time=10522877 us) (object id 7409)

In this sample TKPROF output, note the following under the Row Source Operationcolumn:

■ cr specifies consistent reads performed by the row source

■ r specifies physical reads performed by the row source

■ w specifies physical writes performed by the row source

■ time specifies time in microseconds

To ensure that wait events information is written to the trace file for the session,

Run the following SQL statement:

ALTER SESSION SET EVENTS '10046 trace name context forever, level 8';

**Understanding Recursive Calls**

Sometimes, in order to execute a SQL statement issued by a user, Oracle must issue additional statements. Such statements are called recursive calls or recursive SQL statements. For example, if you insert a row into a table that does not have enough space to hold that row, then Oracle makes recursive calls to allocate the space dynamically. Recursive calls are also generated when data dictionary information is not available in the data dictionary cache and must be retrieved from disk.

If the Parse column showed a large number for the same statement, it would be an indicator that bind variables weren’t being used.

• The Fetch operation was performed 17,324 times and fetched 259,806 rows. Because the

number of rows is far greater than the number of fetches, you can deduce that Oracle used array fetch operations.

There were three physical reads during the fetch operation. If there’s a large difference

between CPU time and elapsed time, it can be attributed to time taken up by disk reads. In

this case, the physical I/O has a value of only 3, and it matches the insignificant gap between

CPU time and elapsed time. The fetch required 136 buffer gets in the consistent mode and only 5 DB block gets.

**Examining the Formatted Output File**

Listing 21-11 shows the top portion of the test.txt file, which explains the key terms used by the utility.

**Listing 21-11.** *The Top Part of the TKPROF-Formatted Trace File*

TKPROF: Release 10.1.0.2.0 - Production on Sat Apr 30 14:42:45 2005

Copyright (c) 1982, 2004, Oracle. All rights reserved.

Trace file: finance\_ora\_16340.trc

Sort options: default

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

count = number of times OCI procedure was executed

cpu = cpu time in seconds executing

elapsed = elapsed time in seconds executing

disk = number of physical reads of buffers from disk

query = number of buffers gotten for consistent read

current = number of buffers gotten in current mode (usually for update)

rows = number of rows processed by the fetch or execute call

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Each TKPROF report shows the following information for each SQL statement issued during the time the user’s session was traced:

• The SQL statement

• Counts of parse, execute, and fetch (for SELECT statements) calls

• Count of rows processed

• CPU seconds used

• I/O used

• Library cache misses

• Optional execution plan

• Row-source operation listing

• A report summary analyzing how many similar and distinct statements were found in the trace file

Let’s analyze the formatted output created by TKPROF. Listing 21-12 shows the parts of the

TKPROF output showing the parse, execute, and fetch counts.

**Listing 21-12.** *The Parse, Execute, and Fetch Counts*

SQL> select e.last\_name,e.first\_name,d.department\_name from teste e,testd d

where e.department\_id=d.department\_id;

call count cpu elapsed disk query current rows

------- ------ ------ ---------- -- ---------- ---------- --------

Parse 1 0.00 0.00 0 0 0 0

Execute 1 0.00 0.00 0 0 0 0

Fetch 17322 1.82 1.85 3 136 5 259806

------- ------ -------- ---------- ---------- ---------- ----------

total 17324 1.82 1.85 3 136 5 259806

Misses in library cache during parse: 0

Optimizer goal: CHOOSE

Parsing user id: 53

In Listing 21-12

• *CPU* stands for total CPU time in seconds.

• *Elapsed* is the total time elapsed in seconds.

• *Disk* denotes total physical reads.

• *Query* is the number of consistent buffer gets.

• *Current* is the number of database block gets.

• *Rows* is the total number of rows processed for each type of call.

From Listing 21-12, you can draw the following conclusions:

• The SQL statement shown previously was parsed once, so a parsed version wasn’t available

in the shared pool before execution. The Parse column shows that this operation took less

than 0.01 seconds. Note that the lack of disk I/Os and buffer gets indicates that there were no

data dictionary cache misses during the parse operation. If the Parse column showed a large

number for the same statement, it would be an indicator that bind variables weren’t being

used.

• The statement was executed once and execution took less than 0.01 seconds. Again, there

were no disk I/Os or buffer gets during the execution phase.

• It took me a lot longer than 0.01 seconds to get the results of the SELECT statement back. The

Fetch column answers this question of why that should be: it shows that the operation was

performed 17,324 times and took up 1.82 seconds of CPU time.

• The Fetch operation was performed 17,324 times and fetched 259,806 rows. Because the

number of rows is far greater than the number of fetches, you can deduce that Oracle used

array fetch operations.

• There were three physical reads during the fetch operation. If there’s a large difference

between CPU time and elapsed time, it can be attributed to time taken up by disk reads. In

this case, the physical I/O has a value of only 3, and it matches the insignificant gap between

CPU time and elapsed time. The fetch required 136 buffer gets in the consistent mode and

only 5 DB block gets.

• The CBO was being used, because the Optimizer goal is shown as CHOOSE.

The following output shows the execution plan that was explicitly requested when TKPROF

was invoked. Note that instead of the cost estimates that you get when you use the EXPLAIN PLAN

tool, you get the number of rows output by each step of the execution.

Rows Row Source Operation

------- -----------------------

259806 MERGE JOIN

1161 SORT JOIN

1161 TABLE ACCESS FULL TESTD

259806 SORT JOIN

Finally, TKPROF summarizes the report, stating how many SQL statements were traced. Here’s

the summary portion of the TKPROF-formatted output:

Trace file: ORA02344.TRC

Trace file compatibility: 9.00.01

Sort options: default

**988** CHAPTER 21 ■ IMPROVING DATABASE PERFORMANCE: SQL QUERY OPTIMIZATION

2 sessions in trace file.

18 user SQL statements in trace file.

104 internal SQL statements in trace file.

72 SQL statements in trace file.

33 unique SQL statements in trace file.

18182 lines in trace file.

The TKPROF output makes it easy to identify inefficient SQL statements. TKPROF can order the

SQL statements by elapsed time (time taken for execution), which tells you which of the SQL statements

you should focus on for optimization.

The SQL Trace utility is a powerful tool in tuning SQL, because it goes far beyond the information

produced by using EXPLAIN PLAN. It provides you with hard information about the number of

the various types of calls made to Oracle during statement execution, and how the resource use was

allocated to the various stages of execution.

The sort keyword is extremely useful. Typically, a TKPROF report may include hundreds of SQL statements, but you may only be interested in a few resource intensive queries. The sort keyword allows you to order the listing of the SQL statements so that you don’t have to scan the entire file looking for resource hogs. In some ways, the sort feature is too powerful for its own good. For example, you cannot sort statements by CPU time consumed—instead you sort by CPU time spent parsing, CPU time spent executing, or CPU time spent fetching.

A sample TKPROF report for the invoice item query we’ve been using so far is as follows:

TKPROF: Release 8.1.6.1.0 - Production on Wed Aug 9 19:06:36 2000

(c) Copyright 1999 Oracle Corporation. All rights reserved.

Trace file: example.trc

Sort options: default

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

count = number of times OCI procedure was executed

cpu = cpu time in seconds executing

elapsed = elapsed time in seconds executing

disk = number of physical reads of buffers from disk

query = number of buffers gotten for consistent read

current = number of buffers gotten in current mode (usu

ally for update)

rows = number of rows processed by the fetch or execute call

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ALTER SESSION /\* TKPROF example \*/ SET sql\_trace = TRUE

call count cpu elapsed disk query current rows

------- ------ -------- ---------- ---------- ---------- ---------- ----------

Parse 0 0.00 0.00 0 0 0 0

Execute 1 0.00 0.00 0 0 0 0

Fetch 0 0.00 0.00 0 0 0 0

------- ------ -------- ---------- ---------- ---------- ---------- ----------

total 1 0.00 0.00 0 0 0 0

Misses in library cache during parse: 0

Misses in library cache during execute: 1

Optimizer goal: CHOOSE

Parsing user id: 34 (RSCHRAG)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ALTER SESSION SET timed\_statistics = TRUE

call count cpu elapsed disk query current rows

------- ------ -------- ---------- ---------- ---------- ---------- ----------

Parse 1 0.00 0.00 0 0 0 0

Execute 1 0.00 0.00 0 0 0 0

Fetch 0 0.00 0.00 0 0 0 0

------- ------ -------- ---------- ---------- ---------- ---------- ----------

total 2 0.00 0.00 0 0 0 0

Misses in library cache during parse: 1

Optimizer goal: CHOOSE

Parsing user id: 34 (RSCHRAG)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SELECT a.customer\_name, a.customer\_number, b.invoice\_number,

b.invoice\_type, b.invoice\_date, b.total\_amount, c.line\_number,

c.part\_number, c.quantity, c.unit\_cost

FROM customers a, invoices b, invoice\_items c

WHERE c.invoice\_id = :b1

AND c.line\_number = :b2

AND b.invoice\_id = c.invoice\_id

AND a.customer\_id = b.customer\_id

call count cpu elapsed disk query current rows

------- ------ -------- ---------- ---------- ---------- ---------- ----------

Parse 1 0.05 0.02 0 0 0 0

Execute 1 0.00 0.00 0 0 0 0

Fetch 2 0.00 0.00 8 8 0 1

------- ------ -------- ---------- ---------- ---------- ---------- ----------

total 4 0.05 0.02 8 8 0 1

Misses in library cache during parse: 1

Optimizer goal: CHOOSE

Parsing user id: 34 (RSCHRAG)

Rows Row Source Operation

------- ---------------------------------------------------

1 NESTED LOOPS

1 NESTED LOOPS

1 TABLE ACCESS BY INDEX ROWID INVOICE\_ITEMS

1 INDEX UNIQUE SCAN (object id 21892)

1 TABLE ACCESS BY INDEX ROWID INVOICES

1 INDEX UNIQUE SCAN (object id 21889)

1 TABLE ACCESS BY INDEX ROWID CUSTOMERS

1 INDEX UNIQUE SCAN (object id 21887)

Rows Execution Plan

------- ---------------------------------------------------

0 SELECT STATEMENT GOAL: CHOOSE

1 NESTED LOOPS

1 NESTED LOOPS

1 TABLE ACCESS GOAL: ANALYZED (BY INDEX ROWID) OF

'INVOICE\_ITEMS'

1 INDEX GOAL: ANALYZED (UNIQUE SCAN) OF 'INVOICE\_ITEMS\_PK'

(UNIQUE)

1 TABLE ACCESS GOAL: ANALYZED (BY INDEX ROWID) OF

'INVOICES'

1 INDEX GOAL: ANALYZED (UNIQUE SCAN) OF 'INVOICES\_PK'

(UNIQUE)

1 TABLE ACCESS GOAL: ANALYZED (BY INDEX ROWID) OF 'CUSTOMERS'

1 INDEX GOAL: ANALYZED (UNIQUE SCAN) OF 'CUSTOMERS\_PK'

(UNIQUE)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ALTER SESSION SET sql\_trace = FALSE

call count cpu elapsed disk query current rows

------- ------ -------- ---------- ---------- ---------- ---------- ----------

Parse 1 0.00 0.00 0 0 0 0

Execute 1 0.00 0.00 0 0 0 0

Fetch 0 0.00 0.00 0 0 0 0

------- ------ -------- ---------- ---------- ---------- ---------- ----------

total 2 0.00 0.00 0 0 0 0

Misses in library cache during parse: 1

Optimizer goal: CHOOSE

Parsing user id: 34 (RSCHRAG)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

OVERALL TOTALS FOR ALL NON-RECURSIVE STATEMENTS

call count cpu elapsed disk query current rows

------- ------ -------- ---------- ---------- ---------- ---------- ----------

Parse 3 0.05 0.02 0 0 0 0

Execute 4 0.00 0.00 0 0 0 0

Fetch 2 0.00 0.00 8 8 0 1

------- ------ -------- ---------- ---------- ---------- ---------- ----------

total 9 0.05 0.02 8 8 0 1

Misses in library cache during parse: 3

Misses in library cache during execute: 1

OVERALL TOTALS FOR ALL RECURSIVE STATEMENTS

call count cpu elapsed disk query current rows

------- ------ -------- ---------- ---------- ---------- ---------- ----------

Parse 24 0.02 0.04 1 0 1 0

Execute 62 0.01 0.05 0 0 0 0

Fetch 126 0.02 0.02 6 198 0 100

------- ------ -------- ---------- ---------- ---------- ---------- ----------

total 212 0.05 0.11 7 198 1 100

Misses in library cache during parse: 11

4 user SQL statements in session.

24 internal SQL statements in session.

28 SQL statements in session.

1 statement EXPLAINed in this session.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Trace file: example.trc

Trace file compatibility: 8.00.04

Sort options: default

1 session in tracefile.

4 user SQL statements in trace file.

24 internal SQL statements in trace file.

28 SQL statements in trace file.

15 unique SQL statements in trace file.

1 SQL statements EXPLAINed using schema:

RSCHRAG.prof$plan\_table

Default table was used.

Table was created.

Table was dropped.

381 lines in trace file.

You can see that there is a lot going on in a TKPROF report. We will talk about how to read the report and interpret the different statistics in the next section.

**Reading TKPROF Reports**

Every TKPROF report starts with a header that lists the TKPROF version, the date and time the report was generated, the name of the trace file, the sort option used, and a brief definition of the column headings in the report. Every report ends with a series of summary statistics. You can see the heading and summary statistics on the sample TKPROF report shown earlier in this paper.

The main body of the TKPROF report consists of one entry for each distinct SQL statement that was executed by the database server while SQL trace was enabled. There are a few subtleties at play in the previous sentence. If an application queries the customers table 50 times, each time specifying a different customer\_id as a literal, then there will be 50 separate entries in the TKPROF report. If however, the application specifies the customer\_id as a bind variable, then there will be only one entry in the report with an indication that the statement was executed 50 times. Furthermore, the report will also include SQL statements initiated by the database server itself in order to perform so-called “recursive operations” such as manage the data dictionary and dictionary cache.

The entries for each SQL statement in the TKPROF report are separated by a row of asterisks. The first part of each entry lists the SQL statement and statistics pertaining to the parsing, execution, and fetching of the SQL statement. Consider the following example:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SELECT table\_name

FROM user\_tables

ORDER BY table\_name

call count cpu elapsed disk query current rows

------- ------ -------- ---------- ---------- ---------- ---------- ----------

Parse 1 0.01 0.02 0 0 0 0

Execute 1 0.00 0.00 0 0 0 0

Fetch 14 0.59 0.99 0 33633 0 194

------- ------ -------- ---------- ---------- ---------- ---------- ----------

total 16 0.60 1.01 0 33633 0 194

Misses in library cache during parse: 1

Optimizer goal: CHOOSE

Parsing user id: RSCHRAG [recursive depth: 0]

This may not seem like a useful example because it is simply a query against a dictionary view and does not involve application tables. However, this query actually serves the purpose well from the standpoint of highlighting the elements of a TKPROF report.

Reading across, we see that while SQL trace was enabled, the application called on the database server to parse this statement once. 0.01 CPU seconds over a period of 0.02 elapsed seconds were used on the parse call, although no physical disk I/Os or even any buffer gets were required. (We can infer that all dictionary data required to parse the statement were already in the dictionary cache in the SGA.)

The next line shows that the application called on Oracle to execute the query once, with less than 0.01 seconds of CPU time and elapsed time being used on the execute call. Again, no physical disk I/Os or buffer gets were required. The fact that almost no resources were used on the execute call might seem strange, but it makes perfect sense when you consider that Oracle defers all work on most SELECT statements until the first row is fetched.

The next line indicates that the application performed 14 fetch calls, retrieving a total of 194 rows. The 14 calls used a total of 0.59 CPU seconds and 0.99 seconds of elapsed time. Although no physical disk I/Os were performed, 33,633 buffers were gotten in consistent mode (consistent gets). In other words, there were 33,633 hits in the buffer cache and no misses. I ran this query from SQL\*Plus, and we can see here that SQL\*Plus uses an array interface to fetch multiple rows on one fetch call. We can also see that, although no disk I/Os were necessary, it took quite a bit of processing to complete this query.

The remaining lines on the first part of the entry for this SQL statement show that there was a miss in the library cache (the SQL statement was not already in the shared pool), the CHOOSE optimizer goal was used to develop the execution plan, and the parsing was performed in the RSCHRAG schema.

Notice the text in square brackets concerning recursive depth. This did not actually appear on the report—I added it for effect. The fact that the report did not mention recursive depth for this statement indicates that it was executed at the top level. In other words, the application issued this statement directly to the database server. When recursion is involved, the TKPROF report will indicate the depth of the recursion next to the parsing user.

There are two primary ways in which recursion occurs. Data dictionary operations can cause recursive SQL operations. When a query references a schema object that is missing from the dictionary cache, a recursive query is executed in order to fetch the object definition into the dictionary cache. For example, a query from a view whose definition is not in the dictionary cache will cause a recursive query against view$ to be parsed in the SYS schema. Also, dynamic space allocations in dictionary-managed tablespaces will cause recursive updates against uet$ and fet$ in the SYS schema.

Use of database triggers and stored procedures can also cause recursion. Suppose an application inserts a row into a table that has a database trigger. When the trigger fires, its statements run at a recursion depth of one. If the trigger invokes a stored procedure, the recursion depth could increase to two. This could continue through any number of levels.

So far we have been looking at the top part of the SQL statement entry in the TKPROF report. The remainder of the entry consists of a row source operation list and optionally an execution plan display. (If the explain keyword was not used when the TKPROF report was generated, then the execution plan display will be omitted.) Consider the following example, which is the rest of the entry shown above:

Rows Row Source Operation

------- ---------------------------------------------------

194 SORT ORDER BY

194 NESTED LOOPS

195 NESTED LOOPS OUTER

195 NESTED LOOPS OUTER

195 NESTED LOOPS

11146 TABLE ACCESS BY INDEX ROWID OBJ$

11146 INDEX RANGE SCAN (object id 34)

11339 TABLE ACCESS CLUSTER TAB$

12665 INDEX UNIQUE SCAN (object id 3)

33 INDEX UNIQUE SCAN (object id 33)

193 TABLE ACCESS CLUSTER SEG$

387 INDEX UNIQUE SCAN (object id 9)

194 TABLE ACCESS CLUSTER TS$

388 INDEX UNIQUE SCAN (object id 7)

Rows Execution Plan

------- ---------------------------------------------------

0 SELECT STATEMENT GOAL: CHOOSE

194 SORT (ORDER BY)

194 NESTED LOOPS

195 NESTED LOOPS (OUTER)

195 NESTED LOOPS (OUTER)

195 NESTED LOOPS

11146 TABLE ACCESS (BY INDEX ROWID) OF 'OBJ$'

11146 INDEX (RANGE SCAN) OF 'I\_OBJ2' (UNIQUE)

11339 TABLE ACCESS (CLUSTER) OF 'TAB$'

12665 INDEX (UNIQUE SCAN) OF 'I\_OBJ#' (NON-UNIQUE)

33 INDEX (UNIQUE SCAN) OF 'I\_OBJ1' (UNIQUE)

193 TABLE ACCESS (CLUSTER) OF 'SEG$'

387 INDEX (UNIQUE SCAN) OF 'I\_FILE#\_BLOCK#' (NON-UNIQUE)

194 TABLE ACCESS (CLUSTER) OF 'TS$'

388 INDEX (UNIQUE SCAN) OF 'I\_TS#' (NON-UNIQUE)

The row source operation listing looks very much like an execution plan. It is based on data collected from the SQL trace file and can be thought of as a “poor man’s execution plan”. It is close, but not complete.

The execution plan shows the same basic information you could get from the autotrace facility of SQL\*Plus or by querying the plan table after an EXPLAIN PLAN statement—with one key difference. The rows column along the left side of the execution plan contains a count of how many rows of data Oracle processed at each step during the execution of the statement. This is not an estimate from the optimizer, but rather actual counts based on the contents of the SQL trace file.

Although the query in this example goes against a dictionary view and is not terribly interesting, you can see that Oracle did a lot of work to get the 194 rows in the result: 11,146 range scans were performed against the i\_obj2 index, followed by 11,146 accesses on the obj$ table. This led to 12,665 non-unique lookups on the i\_obj# index, 11,339 accesses on the tab$ table, and so on.

In situations where it is feasible to actually execute the SQL statement you wish to explain (as opposed to merely parsing it as with the EXPLAIN PLAN statement), I believe TKPROF offers the best execution plan display. GUI tools such as TOAD will give you results with much less effort, but the display you get from TOAD is not 100% complete and in certain situations critical information is missing. (Again, my experience is with the free version!) Meanwhile, simple plan table query scripts like my explain.sql presented earlier in this paper or utlxpls.sql display very incomplete information. TKPROF gives the most relevant detail, and the actual row counts on each operation can be very useful in diagnosing performance problems. Autotrace in SQL\*Plus gives you most of the information and is easy to use, so I give it a close second place.

**TKPROF Reports: More Than Just Execution Plans**

The information displayed in a TKPROF report can be extremely valuable in the application tuning process. Of course the execution plan listing will give you insights into how Oracle executes the SQL statements that make up the application, and ways to potentially improve performance. However, the other elements of the TKPROF report can be helpful as well.

Looking at the repetition of SQL statements and the library cache miss statistics, you can determine if the application is making appropriate use of Oracle’s shared SQL facility. Are bind variables being used, or is every query a unique statement that must be parsed from scratch?

From the counts of parse, execute, and fetch calls, you can see if applications are making appropriate use of Oracle’s APIs. Is the application fetching rows one at a time? Is the application reparsing the same cursor thousands of times instead of holding it open and avoiding subsequent parses? Is the application submitting large numbers of simple SQL statements instead of bulking them into PL/SQL blocks or perhaps using array binds?

Looking at the CPU and I/O statistics, you can see which statements consume the most system resources. Could some statements be tuned so as to be less CPU intensive or less I/O intensive? Would shaving just a few buffer gets off of a statement’s execution plan have a big impact because the statement gets executed so frequently?

The row counts on the individual operations in an execution plan display can help identify inefficiencies. Are tables being joined in the wrong order, causing large numbers of rows to be joined and eliminated only at the very end? Are large numbers of duplicate rows being fed into sorts for uniqueness when perhaps the duplicates could have been weeded out earlier on?

TKPROF reports may seem long and complicated, but nothing in the report is without purpose. (Well, okay, the row source operation listing sometimes isn’t very useful!) You can learn volumes about how your application interacts with the database server by generating and reading a TKPROF report.