

# Performance Tuning

## Oracle Performance

---

### 1. EXPLAIN\_PLAN:

The `EXPLAIN PLAN` statement displays execution plans chosen by the optimizer for `SELECT`, `UPDATE`, `INSERT`, and `DELETE` statements. A statement execution plan is the sequence of operations that the database performs to run the statement.

The **row source tree** is the core of the execution plan. The tree 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 `EXPLAIN PLAN` results let you determine whether the optimizer selects a particular execution plan, such as, nested loops join. The results also help 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.

After the sql script is explained, the `EXPLAIN PLAN` statement inserts rows describing execution plans into `PLAN_TABLE`. `PLAN_TABLE` is the default sample output table.

The `PLAN_TABLE` is automatically created as a public synonym to a global temporary table "`SYS.PLAN_TABLE$`". This temporary table holds the output of `EXPLAIN PLAN` statements for all users but its records are private and can be seen only by the user explaining that sql script. For instance, if both 'sys' user and 'saman'

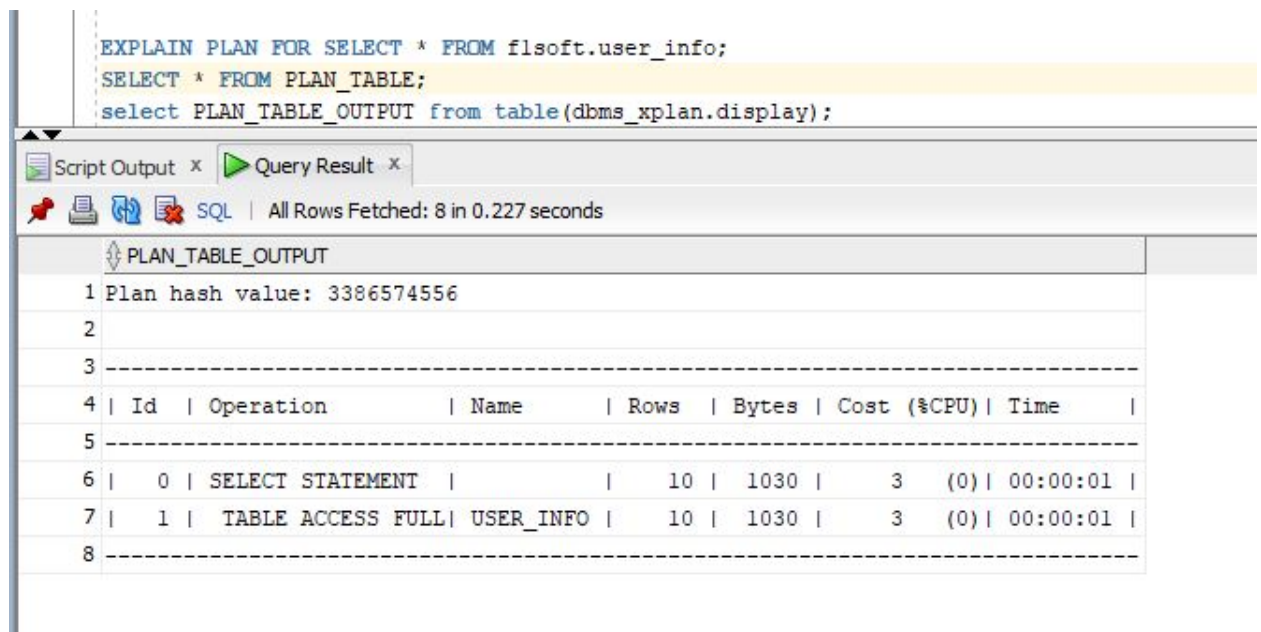
---

user runs the explain script on a query. 'Sys' user sees only execution plan of its query and cannot view the execution plan of other user(Note: even 'sys' user cannot view other users execution plan). And similarly, 'saman' user can see only its execution plan explained on a query not other user's execution plan, despite the fact that plan\_table\$ is a global temporary table.

In order to view the execution plan, we generally use DBMS\_XPLAN.DISPLAY table function.

We could use following options for displaying plan table output.

- Plan table output name
- Statement id
- A format option that determines the level of detail: BASIC, SERIAL, and TYPICAL, ALL



```
EXPLAIN PLAN FOR SELECT * FROM flsoft.user_info;
SELECT * FROM PLAN_TABLE;
select PLAN_TABLE_OUTPUT from table(dbms_xplan.display);
```

PLAN_TABLE_OUTPUT
1 Plan hash value: 3386574556
2
3 -----
4   Id   Operation   Name   Rows   Bytes   Cost (%CPU)   Time
5 -----
6   0   SELECT STATEMENT     10   1030   3 (0)   00:00:01
7   1   TABLE ACCESS FULL   USER_INFO   10   1030   3 (0)   00:00:01
8 -----

Fig 1.1: Explain Plan and view of sql statement execution plan

The above screenshot demonstrate :

- EXPLAIN PLAN on "Select " statement on the records of user\_info table from flsoft user.

- 
- This select statement is explained and added in PLAN\_TABLE\$ table.(Note: PLAN\_TABLE is synonym to PLAN\_TABLE\$ table).
  - Output records are viewed from DBMS\_XPLAN.DISPLAY table function. And the output is seen as shown in above screenshot.
    - ID 0 shows the SELECT statement operation selecting 10 rows of 1030 Bytes with the CPU cost of 3% in 1 sec.
    - ID 1 shows that the USER\_INFO table was fully accessed.

We could also define our own statement ID and plan table using SET STATEMENT\_ID and INTO clause FOR a statement respectively.

For example, we could set 'st1' statement ID into our custom my\_plan\_table PLAN\_TABLE for "*Select last\_name from employee*" statement by following script.

```
EXPLAIN PLAN
SET STATEMENT_ID = 'st1'
INTO my_plan_table
FOR
SELECT last_name FROM employees;
```

We can also view the statistics and execution plan of a query by setting the autotrace on.

```
SQL> SET AUTOTRACE ON EXPLAIN STATISTICS;
```

```

SQL> set autotrace on explain statistics
SQL> select * from demo_tab;

      ID NAME
-----
      1 saman

Execution Plan
-----
Plan hash value: 76495054

-----
| Id | Operation          | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-----
|  0 | SELECT STATEMENT    |           |      1 |      9 |      3 (0) | 00:00:01 |
|  1 |   TABLE ACCESS FULL| DEMO_TAB  |      1 |      9 |      3 (0) | 00:00:01 |
-----

Statistics
-----
      0 recursive calls
      0 db block gets
      7 consistent gets
      0 physical reads
      0 redo size
    596 bytes sent via SQL*Net to client
    520 bytes received via SQL*Net from client
      2 SQL*Net roundtrips to/from client
      0 sorts (memory)
      0 sorts (disk)
      1 rows processed

SQL>

```

Fig 1.2 : Execution Plan from autotrace

We can set the autotrace off using

```
SET AUTOTRACE OFF;
```

## 2. Automatic Performance Statistics:

### 2.1. Overview of Data Gathering:

To effectively diagnose performance problems, we need statistics. Oracle Database generates many types of cumulative statistics for the system, sessions, and individual SQL statements. When analyzing a performance problem, we typically look at the change in statistics (delta value--changes--) over the period you are interested in. Specifically,

---

look at the difference between the cumulative value of a statistic at the start of the period and the cumulative value at the end.

Cumulative values for statistics are generally available in dynamic views like **v\$sysstat** and **v\$sesstat** views. These values are reseted when the database instance is shutdown.

**Automatic Workload Repository (AWR)** records the cumulative and delta values (i.e changes for each snapshot over certain time period) for most of the statistics at all levels except the session level. This process is repeated after certain interval and the result is called **AWR Snapshot**.

There is another type of statistic collected by Oracle Database called **metric**. Basically, A metric is a rate of change in some cumulative statistics. This rate can be measured against variety of units, including time, transactions, or database calls. For instance, the number of database calls per second is a metric. Metric values are also exposed in some V\$ views where the values are the average over some small time interval, typically 60 seconds. Some metric values are also recorded in AWR Snapshots.

A third type of statistical data collected by oracle is sampled data performed by **Active Session History(ASH)** sampler. It samples the current state of all active sessions and database stores this data into memory, where we can access it with a V\$ views.

Another more powerful tool for diagnosing performance problems is the use of **Statistical BASELINE**. A statistical baseline is collection of statistic rates usually taken over time period where the system is performing well at peak load. Comparing statistics captured during a period of bad performance to a baseline helps discover specific statistics that have increased significantly and could be the cause of the problem. AWR supports the capture of baseline data by enabling us to specify and preserve a range of AWR Snapshots as a baseline however, we should be cautious at specifying the time period as the baseline should be of good representation of the peak load of the system.

**Oracle Enterprise Manager** is the recommended tool to view all the data present in the dynamic performance view and history data from the AWR history tables. Enterprise Manager can also be used to capture Operating System and network statistical data that can be correlated with AWR data. To set up Oracle Enterprise Manager: know your host\_name from **v\$instance** view and know your enterprise manager Console HTTP Port

---

from *portlist.ini* file (i.e. 1158 by default). For example, for the host with name *localhost* and port number 1158 type <https://localhost:1158/em>

Some of the statistics are:

## 2.2. Database Statistics:

Database statistics provide information on the *type of load* on the database and *the internal and external resources* used by the database. Some important statistics are:

### *Wait Events:*

Wait events are statistics that are augmented by a server process or thread in order to indicate that it had to wait for an event to finish before being able to continue processing.

Events are grouped into class so that we can perform high level analysis. This class includes : Administrative, Application, Cluster, Commit, Concurrency, Configuration, Idle, Network, Scheduler, System I/O, user I/O, etc... The examples of the waits in some of the classes are:

- Application: locks waits caused by row level locking or explicit lock commands
- Commit: waits for redo log write confirmation after a commit
- Idle: wait events that signify the session is inactive, such as `SQL*Net message from client`
- Network: waits for data to be sent over the network
- User I/O: wait for blocks to be read off a disk

Wait Event for an instance includes statistics for both background and foreground processes. But we are generally concerned with the foreground statistics so we use relevant V\$ views to facilitate tuning.

- `V$SYSTEM_EVENT` view shows wait event statistics for the foreground activities of an instance and the wait event statistics for the instance.
- `V$SYSTEM_WAIT_CLASS` view shows these foreground and wait event instance statistics after aggregating to wait classes.
- `V$SESSION_EVENT` and `V$SESSION_WAIT_CLASS` show wait event and wait class statistics at the session level.

---

As a DBA, we want the WAIT EVENT to be as **less** as possible because it reveals many symptoms problem that might impact performance such as latch contention, buffer contention and I/O contention.

#### *Time Model Statistics:*

Each components has its own set of statistics, when tuning an Oracle Database. The common scale for comparisons of statistics we use Time.

`V$SESS_TIME_MODEL` and `V$SYS_TIME_MODEL` views provide time model statistics.

In time model statistics, we have a component that represent the **total time spent in database calls** and **show the total instance workload**. This component is `DB time`. It is calculated by aggregating the CPU and wait times of all sessions not waiting on idle wait events (i.e. non-idle sessions). `DB time` is measured cumulatively from the time of instance startup from all non-idle user sessions. For example, an instance that has been running for 30 minutes could have four active user sessions whose cumulative `DB time` is approximately 120 minutes.

As a DBA, the objective of tuning an Oracle system is **reducing** `DB time` because we want to reduce the time that users spend in performing some action on the database.

#### *Active Session History:*

Any sessions that is connected to the database and is waiting for an event that does not belong to the idle wait class is considered as an ACTIVE SESSION. This also includes all the session that was on the CPU at the time of sampling. Active sessions are sampled every second and are stored in circular buffer in SGA. The `V$ACTIVE_SESSION_HISTORY` view provides sampled session activity in the instance. The contents of `V$ACTIVE_SESSION_HISTORY` is flushed to disk because the content of this view can get quite large during heavy system activity.

ASH also contains execution plan information for each captured SQL statement. We can use this information to identify which part of SQL execution contributed most to the SQL elapsed time.

---

## 2.3. Operating system Statistics:

### 2.3.1. CPU Statistics:

CPU utilization is the most important Operating System statistics in tuning process.

- The `V$OSSTAT` view captures machine-level information in the database, to determine if hardware-level resource issues exist.
- The `V$SYSMETRIC_HISTORY` view shows a one-hour history of the Host CPU Utilization metric, a representation of percentage of CPU usage at each one-minute interval.
- The `V$SYS_TIME_MODEL` view supplies statistics on the CPU usage by the Oracle database.

Using both sets of statistics enable us to determine whether the Oracle database or other system activity is the cause of the CPU problems.

During the startup of the database, our statistics was



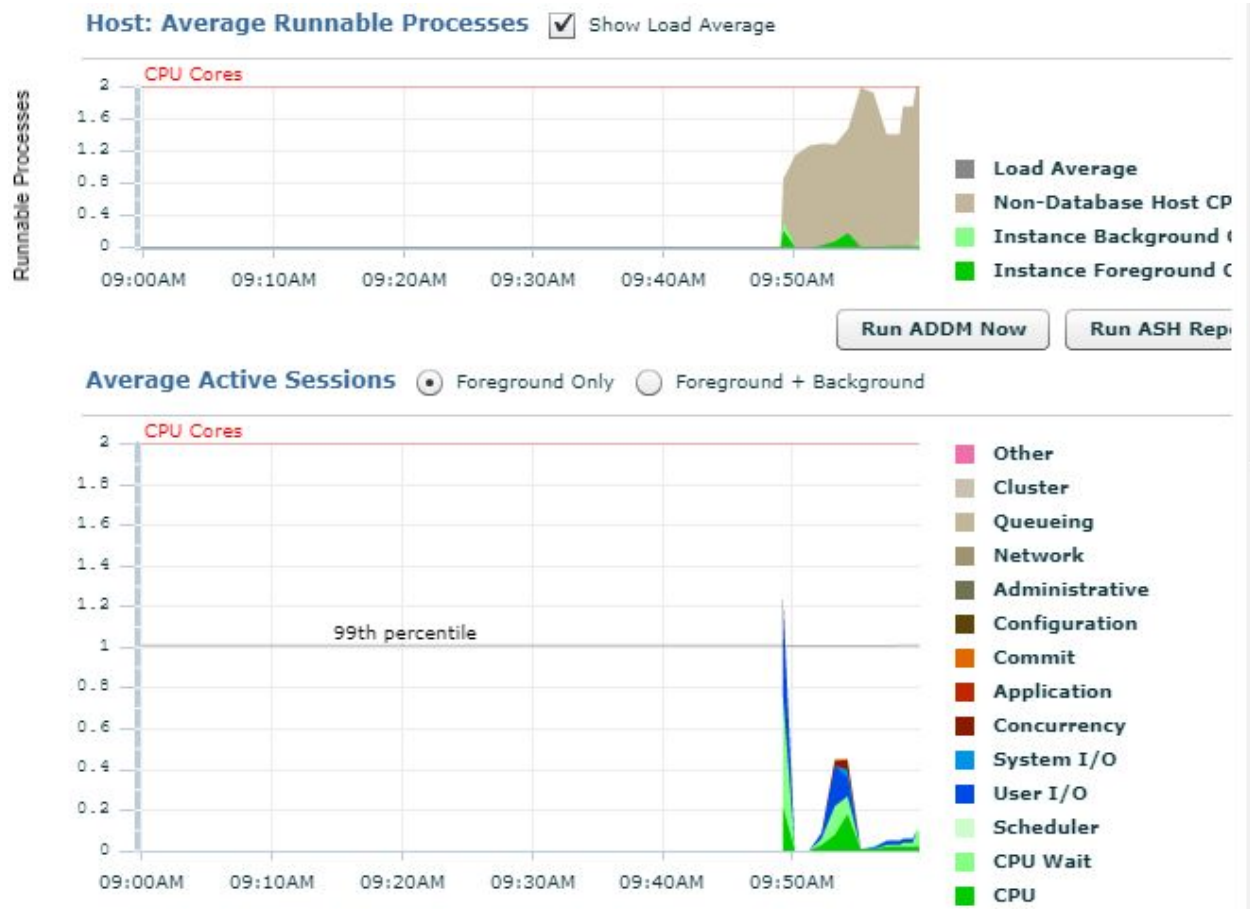


Fig 2.3.1.1: Statistics during startup of database

For instance, we have executed an infinite loop from sys user at 12:15pm using command.

```
SQL> begin
2  while (true)
3  loop
4  NULL;
5  end loop;
6  end;
7  /
```

Fig 2.3.1.2: Infinite Loop

By executing this infinite loop, we viewed its statistics in Oracle Enterprise Manager, our CPU usage increased from 0 to 80 percentile in about 10 minutes.

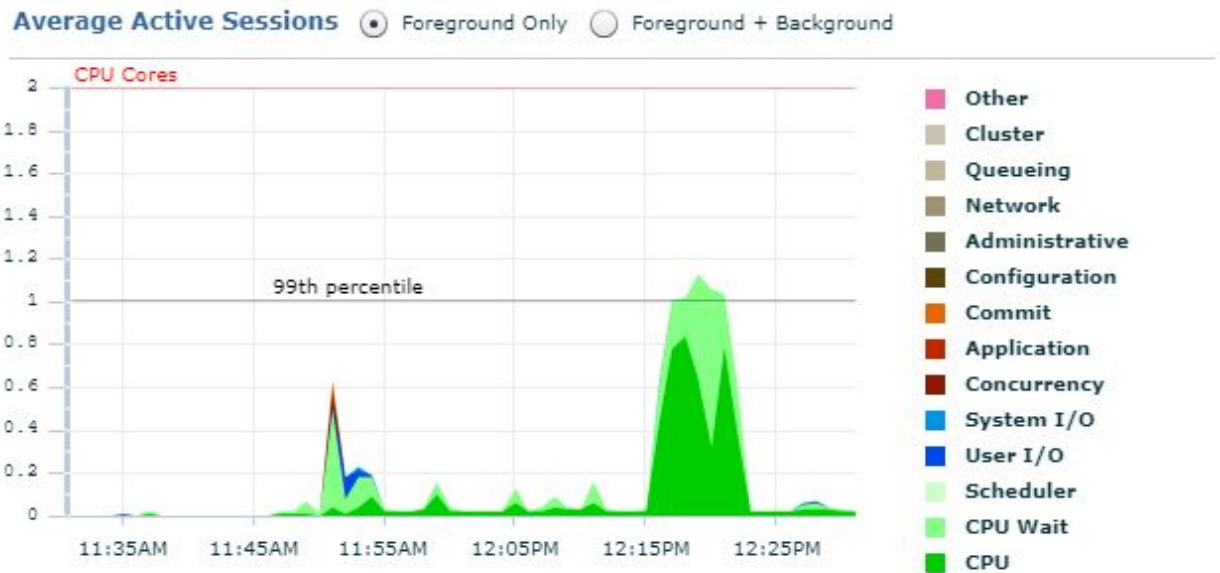


Fig 2.3.1.3: Active Session CPU Usage(in Dark Green)

Also the runnable process for the instance foreground CPU also increased from 0 to 80 percentile. Also notice, CPU wait( light green) is less than CPU usage( dark green). CPU wait greater than CPU is **bad**.

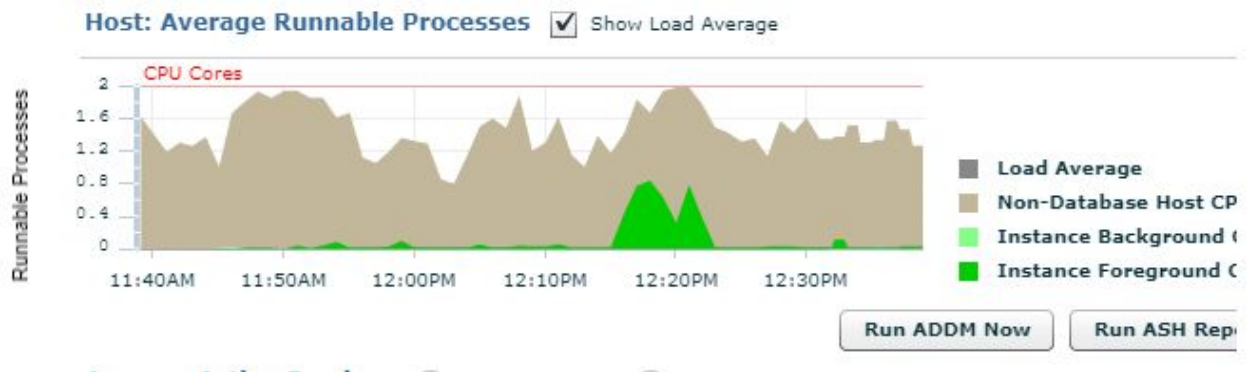


Fig 2.3.1.4: Average Runnable Process VS Instance Foreground CPU

And, Since the infinite loop was executed by 'sys' user, so the service of SYS\$USERS session increased over 99th percentile.



Fig 2.3.1.5: Services statistics while executing infinite loop

We could further have insight over the statistics by clicking **Top Activity** present in the bottom right position at AVERAGE ACTIVE SESSION graph.

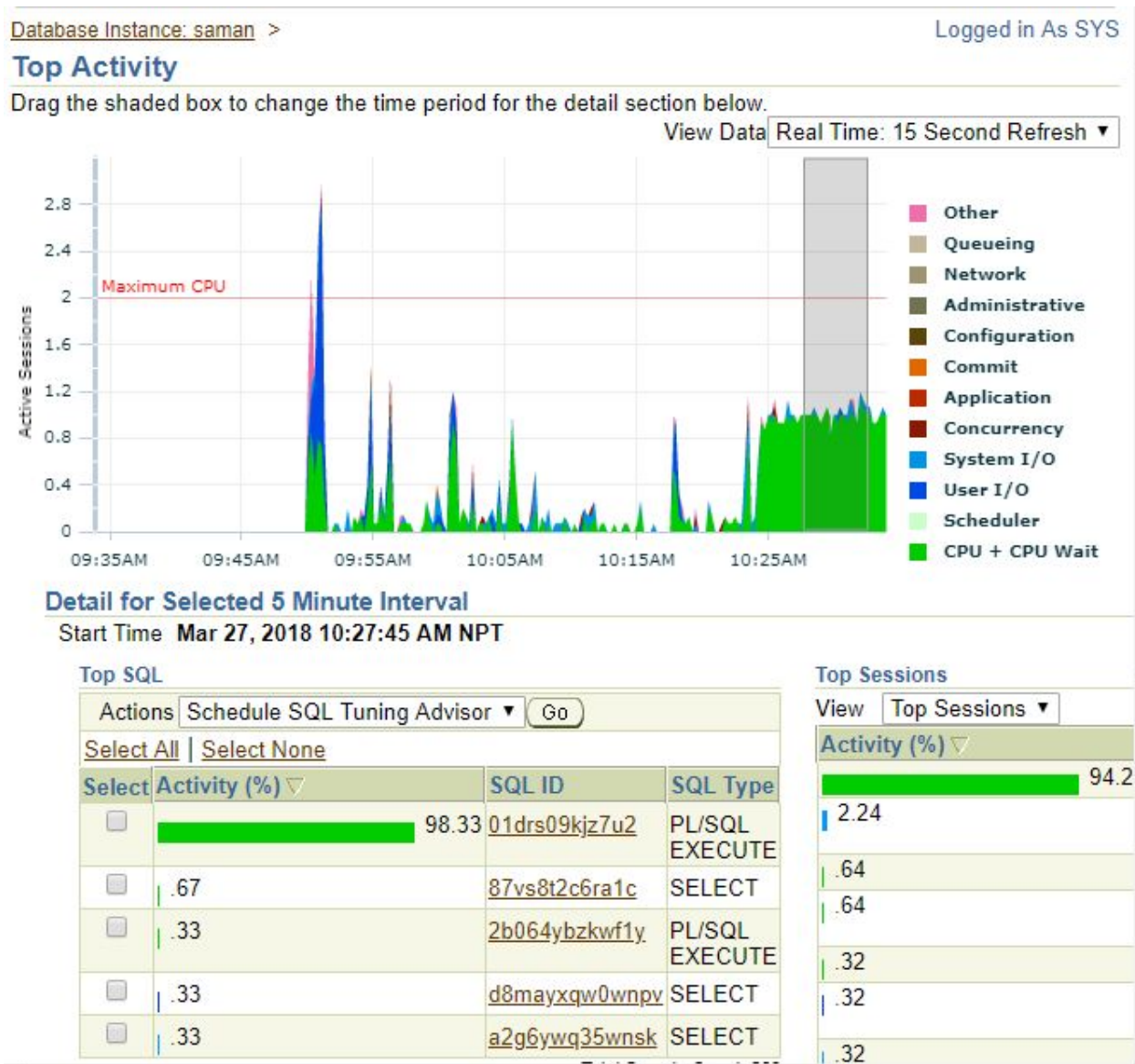


Fig 2.3.1.6: Top Activity

In the top activity, we can see 98.33th percentile CPU is used by *01drs09kz7u2* sql id which is a PL/SQL EXECUTE sql type( which is our infinite loop).

By clicking on that SQL\_ID we get further insights about the CPU usage.

## SQL Details: 01drs09kjj7u2

Switch to SQL ID   View Data

[Text](#) 

```
begin
while(true)
loop
null;
end loop;...
```

### Details

Select the plan hash value to see the details below. Plan Hash Value

[Statistics](#)

**Activity**

[Plan](#)

[Plan Control](#)

[Tuning History](#)

[SQL Monitoring](#)

### Summary

Drag the shaded box to change the time period for the detail section below.



Fig 2.3.1.7: Activity tab of the infinite loop

We can look in the statistics tab too

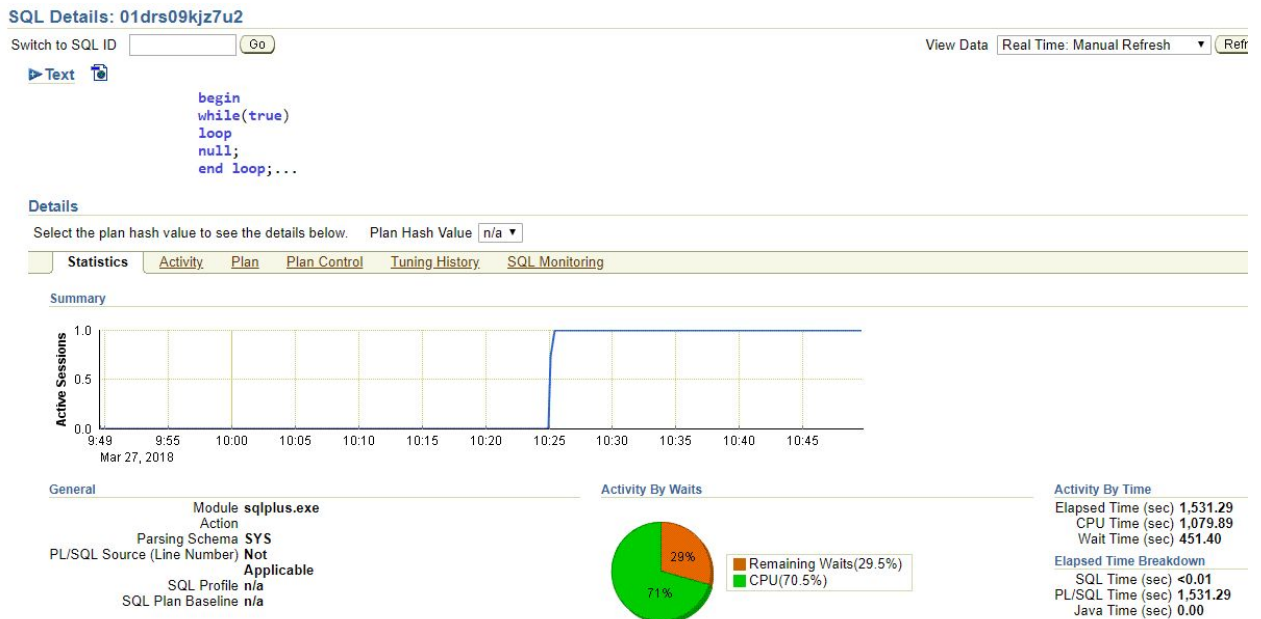


Fig2.3.1.8: Statistics Tab of infinite loop

## 2.4. Disk I/O Statistics:

The most important disk statistics are the current response time and the length of the disk queues. These statistics show if the disk is performing optimally or if the disk is being overworked.

If the hardware shows response times much higher than the normal performance value, then it is performing badly or is overworked. This is your bottleneck.

Oracle Database also maintains a consistent set of I/O statistics for the I/O calls it issues.

For instance, at 1:45pm, I took a RMAN backup and the I/O function showed the following statistics.

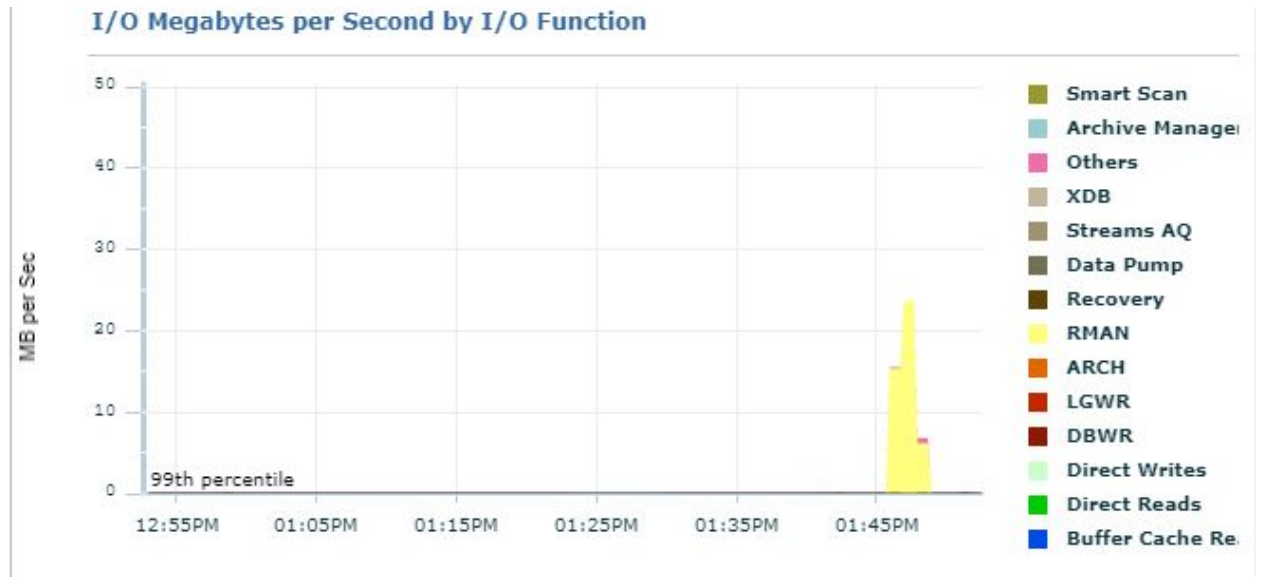


Fig 2.4.1: I/O file read MB per seconds Statistics when performing RMAN Backup

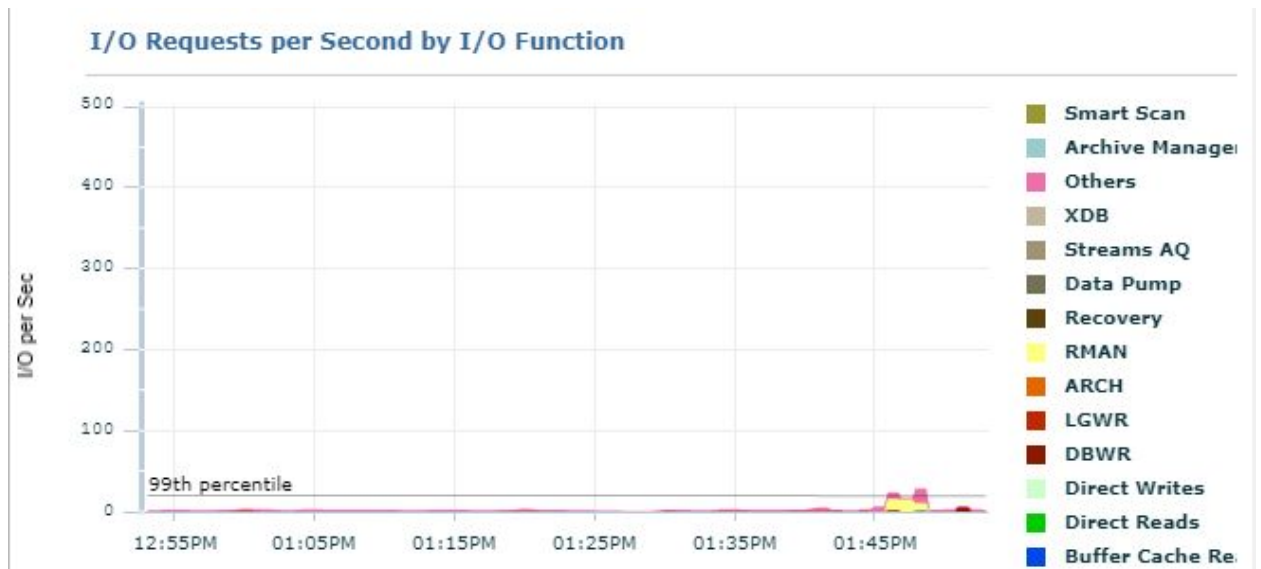


Fig 2.4.2: I/O Request Statistics per second when performing RMAN backup

### 3. Automatic Workload Repository:



---

The Automatic Workload Repository(AWR) collects, processes and maintains performance statistics for problem detection and self-tuning purposes. The data is both in memory and stored in the database.

#### **4. Snapshot:**

Snapshots are sets of historical data for specific time periods that are used for performance comparisons by ADDM(Automatic Diagnostic Database Monitor). 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.

AWR compares the difference between snapshots to determine which SQL statements to capture based on the effect on the system load. This reduces the number of SQL statements that must be captured over time.

We use Oracle Enterprise Manager to view all the statistics and other details including creating, managing, dropping of snapshot but we could also use [DBMS\\_WORKLOAD\\_REPOSITORY](#) procedures to manually create, drop, and modify the snapshots. But To invoke these procedures, a user must be granted the DBA role.

#### **5. Baselines:**

A baseline contains performance data from a specific time period that is preserved for comparison with other similar workload periods when performance problems occur. The snapshots contained in a baseline are excluded from the automatic AWR purging process and are retained indefinitely.

We use Oracle Enterprise Manager to view all the statistics and other details including creating, managing, dropping of snapshot but we could also use [DBMS\\_WORKLOAD\\_REPOSITORY](#) procedures to manually create, drop, and modify the Baselines. But To invoke these procedures, a user must be granted the DBA role.

#### **6. Space Consumption:**

The space consumption by the AWR depends upon many factors like:

- Number of the active sessions in the system at a given time.



- 
- Snapshot interval: If the snapshot interval decreases, then more numbers of snapshot are taken which increases the data collected in the AWR.
  - Historical data retention period: If the retention period is extended, then longer the data is retained before it is purged.

## **7. Automatic Database Diagnostic Monitor(ADDM):**

After the statistics data are stored in Automatic Workload Repository(AWR), we need to analyze the data to perform accurate and timely diagnosis of the problem. Automatic Database Diagnostic Monitor(ADDM) diagnoses the root causes for the performance problem, automatically provides the recommendation, and generates the report of the snapshot. We could use the reports to compare the snapshots taken at different time interval to analyze the SQL Query contributing in the overhead and perform suitable actions good for system

We can use Oracle Enterprise Manager to view the reports of user DBMS\_ADDM.GET\_REPORT function.

```
DBMS_ADDM.GET_REPORT (
    task_name          IN VARCHAR2
    RETURN CLOB) ;
```

And to display the task name

```
SELECT DBMS_ADDM.GET_REPORT(task_name) FROM DUAL;
```

## **References:**

- [1]. [https://docs.oracle.com/cd/E11882\\_01/server.112/e41573/autostat.htm#PFGRF94184](https://docs.oracle.com/cd/E11882_01/server.112/e41573/autostat.htm#PFGRF94184)