

## Oracle Response Time: Optimization with RPM (Response time Profiling Method)

Presented by Ric Van Dyke

[ric.van.dyke@hotsos.com](mailto:ric.van.dyke@hotsos.com)

*Presented: 30 November 2011*

*Hotsos Enterprises, Ltd.  
Grapevine, Texas USA*



**Oracle. Performance. Now.**

## Agenda

- Traditional Tuning Methods
- Using RPM (Target, Collect, React)
- A brief history of extended SQL trace data
- How to get trace data
- Reading the Trace Data
- Using the **BINDS** and **STAT** lines
- The common **WAIT** events
- Q&A

## Hotsos the company...

---

- **Response-Time Profiling Method (RPM)**
  - Focus is on the business
  - Measures exact user experience
  - Trail and error tuning obsolete
- **Products**
  - Hotsos Profiler
  - Laredo
  - HAWCS
- **Education**
  - Oracle performance curriculum
  - Hotsos Symposium
- **Services**
  - On-site consulting and education
  - Remote consulting



## Traditional Tuning Methods

---

## Conventional tuning methods

---

Method	Problems with the method
“Seek utilization statistics (hit ratios) that look ‘wrong’”	<ul style="list-style-type: none"><li>• What if the ugly statistics are irrelevant?</li></ul>
“Tune every layer”	<ul style="list-style-type: none"><li>• Why tune layers that aren’t contributing to your problem?</li></ul>
“Tune the bad SQL first”	<ul style="list-style-type: none"><li>• What if the performance problem is between SQL statements?</li><li>• What if the bad SQL is irrelevant?</li></ul>

Don't waste resources fixing anything other than the thing that is most relevant for your business.

## What traditional tuning methods are missing

---

1. Task focus
  - Can't extrapolate detail from an aggregate (“red rock” problem)
2. Response time focus
  - Can't tell how long something took by counting the number of times it happened
3. Amdahl's Law
  - Improve the biggest response time consumer first
4. Subsystem optimization does not imply system optimization
  - Optimal subsystems can interact suboptimally

## Introduction to RPM

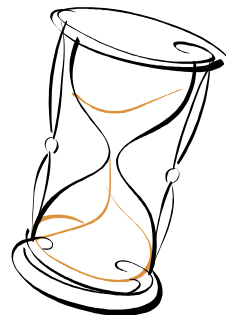
---



## The Golden Rule of Performance Optimization...

---

**Work first to reduce the  
biggest response time  
component  
of the business's most  
important task.**

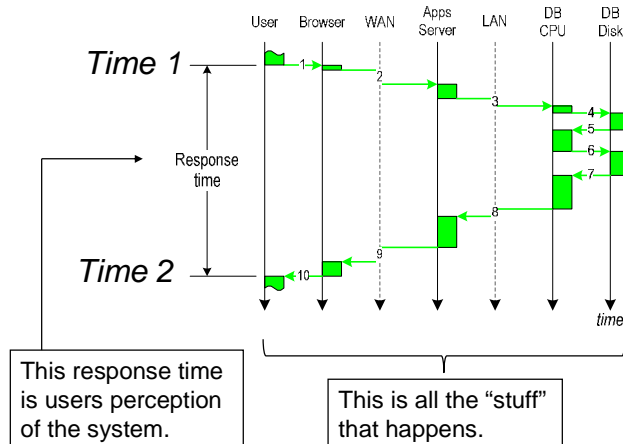


## RPM steps



1. **Target** the tasks for which the business needs improvement.
2. **Collect** properly scoped diagnostic data while the tasks are being slow.
3. **React** with the candidate repair that will have the greatest net payoff to the business.
  - Stop if the cost of the repair exceeds the cost of the problem.
4. Go to step 1 if needed.

## Response time includes time on several layers



## Collect the right data

---

*Perhaps the most common cause of failed performance improvement projects:  
bad data collection.*

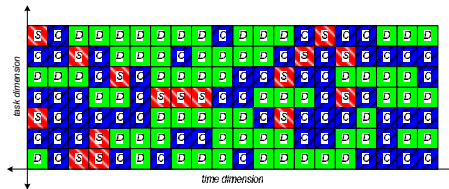
- Two dimensions
  - Right task
  - Right time interval
- Too much data, and you bury the relevant stuff
- Too little data, and you miss stuff entirely

## Targeting the Right Task and Time

---

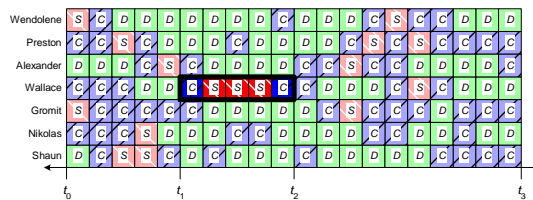


## A two-dimensional picture of your system



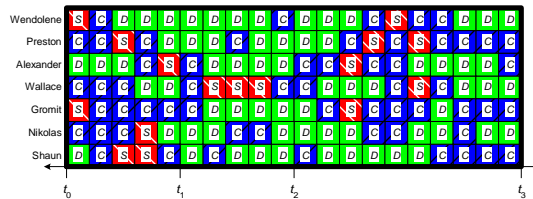
- Horizontal dimension: **time**
- Vertical dimension: **individual tasks**
- Cell contents: resource consumption
  - C – time spent consuming CPU
  - D – time spent consuming disk
  - S – time spent waiting for serialized access to something

## Wallace's program is a critical business task



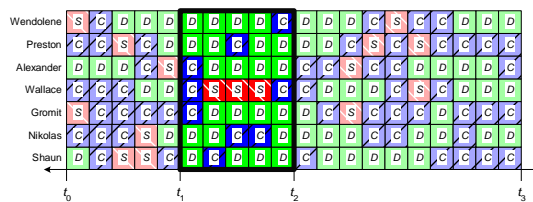
Resource	Elapsed time	
S	3	60.0%
C	2	40.0%
D	0	0.0%
<b>Total</b>	<b>5</b>	<b>100.0%</b>

The easiest way to ruin the project is to scope incorrectly on both dimensions.



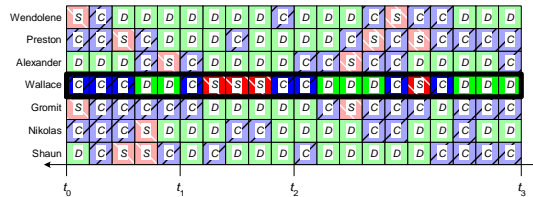
Resource	Elapsed time	
D	66	47.1%
C	58	41.4%
S	16	11.4%
<b>Total</b>	<b>140</b>	<b>100.0%</b>

Even if you correctly scope the time dimension, messing up the task scope can ruin the project.



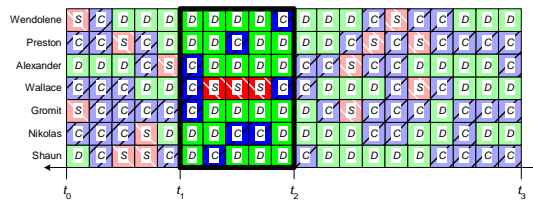
Resource	Elapsed time	
D	23	65.7%
C	9	25.7%
S	3	8.6%
<b>Total</b>	<b>35</b>	<b>100.0%</b>

Even if you correctly scope the task dimension, messing up the time scope can ruin the project.



Resource	Elapsed time	
D	8	40.0%
C	8	40.0%
S	4	20.0%
<b>Total</b>	<b>20</b>	<b>100.0%</b>

“Tuning” the wrong thing can make performance even worse.



- Clearly, *D* is the “system’s bottleneck” for  $[t_1, t_2]$
- Imagine reducing *D* duration for each user
  - More total *C* and *S* workload will move into  $[t_1, t_2]$  interval

Now Wallace’s problem is worse because he waits for *C* and *S* access

Collect the trace data

---



Extended SQL trace data (10046)

---

- Event 10046 instructs the Oracle kernel to emit a sequential record of what it does with your time.
- Event 10046: enable SQL statement timing
  - More accurate: “enable database and system call timing”
- Sequential record logged to a trace file
  - Database calls
  - Oracle “timed events”, if you ask for them
  - Placeholder-value bindings, if you ask for them

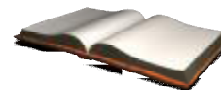
*This is exactly what you need for measuring  
Oracle application response time with precision*

## Brief history ...



- In version 5
  - Internal-only: explicitly not documented and not supported
  - No timing information
- In version 6
  - One line per database call: **PARSE**, **EXEC**, **FETCH**
  - One line per commit or rollback: **XCTEND**
  - One line per row source operation: **STAT**
  - Trace file formatting tool: **tkprof**
- Version 7
  - **WAIT** events add, @100 to start with
  - Example: **WAIT #92: nam='db file sequential read' ela= 0 p1=72 p2=2755 p3=1**

## ... Brief history ...



- Version 8
  - **DBMS\_SUPPORT** package (back ported to 7.2)
  - **WAIT** events up to about 200
- Version 9
  - Time statistics in .000001-second units instead of .01-second
  - **tkprof** parses the **WAIT** lines
  - "STAT" lines response time information
  - Example: **STAT #1 id=4 cnt=1 pid=3 pos=1 obj=130258 op='TABLE ACCESS FULL INVOICE (cr=567 pr=390 pw=0 time=59740 us)'**
  - **WAIT** events up to about 300

## ... Brief history



- Version 10
  - **DBMS\_MONITOR**
  - The **P1**, **P2** and **P3** entries in the **WAIT** events are now translated to the names.
  - Object Number and time stamp added to wait events.
  - Example: `WAIT #3: nam='db file sequential read' ela= 26677 file#=4 block#=233235 blocks=1 obj#=67560 tim=93682960858`
  - **WAIT** events up to about 500
- Version 11
  - **STAT** lines are now written out at the end of a cursor by default.
  - Added on the cost, size, and cardinality.
  - Example: `STAT #3 id=1 cnt=1 pid=0 pos=1 obj=0 op='MERGE JOIN OUTER (cr=4 pr=0 pw=0 time=0 us cost=2 size=189 card=1)'`
  - **WAIT** events up to about 1,100

## Oracle packages



- Event 10046 is so useful and so popular that Oracle Corporation has evolved its interface.
- **DBMS\_SUPPORT**
  - Versions 7.2 and above
  - Works great; “unofficially” supported
    - Run `/rdbms/admin/dbmssupp.sql` as **SYS** to install
- **DBMS\_MONITOR**
  - Version 10 and above
  - Fully supported and documented
    - <http://www.oracle.com/technology/documentation/database10g.html>

## Version 10 example: Using extended SQL trace...

```

$ sqlplus
...
SQL> show parameter...
NAME                                TYPE                                VALUE
-----                                -                                -
timed_statistics                     boolean                             TRUE
max_dump_file_size                   string                             UNLIMITED
user_dump_dest                       string                             /u01/app/oracle/admin/v10/udump
statistics_level                     string                             ALL
...
SQL> exec dbms_monitor.session_trace_enable(null,null,true,true);
SQL> select 'hello' from dual;
SQL> exec dbms_monitor.session_trace_disable(null,null);
...
$ ls -lt $UDUMP
total 248
-rw-r----- 1 oracle oinstall      2396 Jan 17 11:18 v10_ora_5286.trc
...

```

## Version 11 example: Using extended SQL trace...

```

SQL> show parameter...
NAME                                TYPE                                VALUE
-----                                -                                -
timed_statistics                     boolean                             TRUE
max_dump_file_size                   string                             unlimited
diagnostic_dest                     string                             /opt/oracle
user_dump_dest                       string                             /opt/oracle/diag/rdbms/o11/o11/trace
statistics_level                     string                             ALL
...
SQL> exec dbms_monitor.session_trace_enable(null,null,true,true,
'all_executions');
SQL> select 'hello' from dual;
SQL> exec dbms_monitor.session_trace_disable(null,null);
...
$ ls -lt $UDUMP
total 248
-rw-r----- 1 oracle oinstall      3071 May 21 17:09 o11_ora_11754.trc
...

```

## Ending the trace

---

- If you have a dedicated connection instead of disabling trace, just disconnect
- Turning off trace before cursor close withholds the cursor's execution plan information
  - **STAT** lines (Until version 11)
- In today's environments you will likely have to explicitly deactivate tracing
  - Web services
  - Oracle E-Business concurrent manager
  - Connection pooling

## Proper environment settings

---

- `timed_statistics = true`
  - Bad: zero time values in trace data and fixed views
  - Use **TRUE** instance-wide; a necessary function of any application
- `statistics_level = all`
  - Bad: invalid **STAT** time values and parent roll-ups in trace data
  - Use **TRUE** instance-wide; a necessary function of any application
- `max_dump_file_size = unlimited`
  - Bad: **\*\*\* DUMP FILE SIZE IS LIMITED TO n BYTES \*\*\***
  - Use **UNLIMITED** instance-wide; manage disk space another way
- `user_dump_dest = directory-on-big-fast-filesystem`
  - Bad: Slow trace-file writing that holds up your task
  - Bad: **WAIT #42: nam='db file se'**
- `background_dump_dest = directory-on-big-fast-filesystem`
  - Same issues

## TRACEFILE\_IDENTIFIER

---

This parameter is used to identify your trace file by a name you supply.

```
SQL> alter session set tracefile_identifier='MY_TRACE';
SQL> exec dbms_monitor.session_trace_enable(null,null,true,true);
SQL> select 'hello' from dual;
SQL> exit

C:\app\hotsos\diag\rdbms\hotsos\hotsos\trace>dir *my_trace.trc

05-Jul-11  10:42                41,007 hotsos_ora_4748_MY_TRACE.trc
```

## An example of dynamically setting TRACEFILE\_IDENTIFIER

---

```
declare
  t_trc_file varchar2(256) := 'alter session set tracefile_identifier='
    ||chr(39)||to_char(sysdate, 'hh24miss') || '_' || user || chr(39);
  t_trc_stat varchar2(256) := 'alter session set timed_statistics=true';
  t_trc_size varchar2(256) :=
    'alter session set max_dump_file_size=unlimited';
  t_trc_sql varchar2(256) := 'alter session set events ' || chr(39) ||
    '10046 trace name context forever, level 12' || chr(39);
begin
  execute immediate t_trc_file; -- Set tracefile identifier
  execute immediate t_trc_stat; -- Turn on timed statistics
  execute immediate t_trc_size; -- Set max Dump file size
  execute immediate t_trc_sql; -- Turning on Trace
end;
/
```

## Extended SQL Trace Data

- Provides the detail of everything that occurred
- Trace file contents
  - Preamble
  - Session id and timestamp
  - Database calls (**PARSE**, **EXEC**, **FETCH**) for any **SQL**
  - Values bound into **BIND** variables
  - Several **WAIT** lines (interspersed between database calls)
  - Sets of **STAT** lines showing the execution plan for the **SQL**
  - A **CLOSE** line showing when each cursor closes

## Raw Extended SQL Trace Data

```
PARSING IN CURSOR #247467080 len=17 dep=0 uid=91 oct=3 lid=91 tim=203197502895
                    hv=1745700775 ad='270a32ac' sqlid='a2dk8bdn0ujx7'
select * from emp
END OF STMT
PARSE #247467080:c=0,e=80,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,plh=3956160932,
    tim=203197502891
EXEC #247467080:c=0,e=56,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,plh=3956160932,
    tim=203197503359
WAIT #247467080: nam='SQL*Net message to client' ela= 4 driver id=1111838976 #bytes=1
    p3=0 obj#=76115 tim=203197503526
WAIT #247467080: nam='db file sequential read' ela= 12395 file#=4 block#=283066
    blocks=1 obj#=76318 tim=203197516121
WAIT #247467080: nam='db file scattered read' ela= 523 file#=4 block#=283067 blocks=5
    obj#=76318 tim=203197516991
FETCH #247467080:c=0,e=13509,p=6,cr=6,cu=0,mis=0,r=1,dep=0,og=1,plh=3956160932,
    tim=203197517160
WAIT #247467080: nam='SQL*Net message from client' ela= 349 driver id=1111838976
    #bytes=1 p3=0 obj#=76318 tim=203197517647
WAIT #247467080: nam='SQL*Net message to client' ela= 3 driver id=1111838976 #bytes=1
    p3=0 obj#=76318 tim=203197517864
FETCH #247467080:c=0,e=230,p=0,cr=1,cu=0,mis=0,r=13,dep=0,og=1,plh=3956160932,
    tim=203197518067
STAT #247467080 id=1 cnt=14 pid=0 pos=1 obj=76318 op='TABLE ACCESS FULL EMP (cr=7
    pr=6 pw=0 time=13539 us cost=3 size=560 card=14)'
WAIT #247467080: nam='SQL*Net message from client' ela= 5201 driver id=1111838976
    #bytes=1 p3=0 obj#=76318 tim=203197523536
PARSE #245856228:c=0,e=58,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,plh=0, tim=203197524004
```

## Parsing in Cursor

```
PARSING IN CURSOR #247467080 len=17 dep=0 uid=91 oct=3 lid=91
tim=203197502895 hv=1745700775 ad='270a32ac' sqlid='a2dk8bdn0ujx7'
```

Characteristic	Description
len	Length of the statement in bytes
dep	Dependency or Depth
uid	User ID of who ran the statement
oct	Oracle Command Type
lid	Privileged user ID
tim	Time stamp
hv	Hash value of the statement
ad	Address in the library cache
sqlid	The SQLID for the statement

## Database Calls (parse, execute and fetch)

```
PARSE #247467080: c=0, e=80, p=0, cr=0, cu=0, mis=0, r=0, dep=0, og=1,
plh=3956160932, tim=203197502891
```

Characteristic	Description
c	Time spent on the CPU
e	Elapsed (wall clock) time
cr, cu	Consistent and Current reads
mis	Misses on the library cache
r	The number of rows returned
dep	Dependency or depth
og	Optimizer goal
plh	Parse hash value (new in 11.2)
tim	Time stamp



## Wait events

```
WAIT #247467080: nam='db file scattered read' ela= 523 file#=4
block#=283067 blocks=5 obj#=76318 tim=203197516991
```

Characteristic	Description
nam	The name of the event (v\$event_name.name)
ela	Elapsed (wall clock) time
p1, p2, p3	Different for each event
obj#	Object number (DBA_OBJECTS.OBJECT_ID)
tim	Time stamp

## BINDS (more detail on this in a few slides)

```
PARSING IN CURSOR #347277688 len=75 dep=0 uid=92 oct=3 lid=92
tim=113837526275 hv=3452889084 ad='7ff4f607348' sqlid='5zpagur6wxtzw'
select * from employees
where first_name = :name and commission_pct = :comp
END OF STMT
PARSE #347277688:c=0,e=134,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,
plh=1445457117,tim=113837526274
```

```
BINDS #347277688:
```

```
Bind#0
```

```
oacdty=01 mxl=32(10) mxlc=00 mal=00 scl=00 pre=00
oacflg=03 fl2=1000000 frm=01 csi=178 siz=56 off=0
kxsbbbfp=14b31788 bln=32 avl=05 flg=05
value="David"
```

```
Bind#1
```

```
oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
oacflg=03 fl2=1000000 frm=00 csi=00 siz=0 off=32
kxsbbbfp=14b317a8 bln=22 avl=00 flg=01
```

```
EXEC #347277688:c=0,e=280,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,
plh=1445457117,tim=113837526817
```

## STAT lines

```
STAT #247467080 id=1 cnt=14 pid=0 pos=1 obj=76318 op='TABLE ACCESS FULL
EMP (cr=7 pr=6 pw=0 time=13539 us cost=3 size=560 card=14)'
```

Characteristic	Description
id	The step ID
cnt	The number of rows "worked with" for this step
pid	The id of the parent of this step
obj	Object number (DBA_OBJECTS.OBJECT_ID)
op	The row source operation and stats of that step
cr	LOIs for the step
pr, pw	Physical reads and physical writes
time	Elapsed time of step included children
cost, size, card	Same as from the explain plan

## The STAT Lines And The Explain Plan

```
STAT #358786992 id=1 cnt=3 pid=0 pos=1 obj=0 op='SORT GROUP BY NOSORT
(cr=6 pr=0 pw=0 time=178 us cost=1 size=12 card=2)'
```

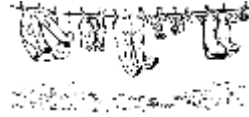
```
STAT #358786992 id=2 cnt=13 pid=1 pos=1 obj=0 op='NESTED LOOPS
(cr=6 pr=0 pw=0 time=140 us cost=1 size=78 card=13)'
```

```
STAT #358786992 id=3 cnt=4 pid=2 pos=1 obj=75994 op='INDEX FULL SCAN
DEPT_DEPTNO_PK (cr=2 pr=0 pw=0 time=39 us cost=1 size=12 card=4)'
```

```
STAT #358786992 id=4 cnt=13 pid=2 pos=2 obj=76000 op='INDEX RANGE SCAN
EMP_DEPT_IDX (cr=4 pr=0 pw=0 time=75 us cost=0 size=9 card=3)'
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)
0	SELECT STATEMENT		2	12	1 (0)
1	SORT GROUP BY NOSORT		2	12	1 (0)
2	NESTED LOOPS		13	78	1 (0)
3	INDEX FULL SCAN	DEPT_DEPTNO_PK	4	12	1 (0)
* 4	INDEX RANGE SCAN	EMP_DEPT_IDX	3	9	0 (0)

## The CLOSE Line



```
CLOSE #247467080:c=0,e=18,dep=0,type=1,tim=203197526828
```

- This is when the cursor is closed
- New trace line in 11
- The **TYPE** field has the following possible values
  - 0: hard close; the cursor is not put into the server-side cursor cache
  - 1: the cursor is cached in a previously empty slot of the server-side cursor cache, since it executed at least 3 times
  - 2: the cursor is placed in a slot of the server-side cursor cache at the expense of aging out another cursor, since it executed at least 3 times
  - 3: the cursor remains in the server-side cursor cache

## How the **BINDS** and **STAT** lines help us with SQL tuning

## SQL Optimization

- Of particular interest in SQL optimization is the **BINDS** and the set of **STAT** lines.
- **BINDS** These lines show what values were used for the bind variables in the statement.
  - Lets you easily build test cases for complex statements.
  - Allows you to see if there is different performance for different input values.
- **STAT** These lines show exactly what the Oracle did do, not what it "might" do.
  - These are lines of the actual plan used.
  - They show resource usage and how long each step took.

**BINDS** lines show you the values of the placeholder variables used by the cursor.

```

PARSING IN CURSOR #347277688 len=75 dep=0 uid=92 oct=3 lid=92
  tim=113837526275 hv=3452889084 ad='7ff4f607348' sqlid='5zpagur6wxtzw'
select * from employees
where first_name = :name and commission_pct = :comp
END OF STMT
PARSE #347277688:c=0,e=134,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,
  plh=1445457117,tim=113837526274
BINDS #347277688:
Bind#0
  oacdty=01 mxl=32(10) mxlc=00 mal=00 scl=00 pre=00
  oacflg=03 fl2=1000000 frm=01 csi=178 siz=56 off=0
  kxsbbbfp=14b31788 bln=32 avl=05 flg=05
  value="David"
Bind#1
  oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
  oacflg=03 fl2=1000000 frm=00 csi=00 siz=0 off=32
  kxsbbbfp=14b317a8 bln=22 avl=00 flg=01
EXEC #347277688:c=0,e=280,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,
  plh=1445457117,tim=113837526817
    
```

## Useful statistics in the BINDS lines

- **bind N** - The bind position within the statement being bound. The first one is 0 (zero).
- **dt** (**oacdt** in 11) - Datatype. This value relates to each datatype and there are both internal and external datatypes. For example some of the internal datatypes are: 1 is **VARCHAR2** and **NVARCHAR2**, 2 is **NUMBER** and 12 is **DATE**. (See the *Oracle Call Interface Programmer's Guide* for a complete list of the values.)
- **avl** - Actual value length or array value length.
- **value** - The actual value of the bind variable.

Metalink note 39817.1, "Interpreting Raw SQL\_TRACE and DBMS\_SUPPORT.START\_TRACE output" covers these and the other statistics.

## Reading a stat line is a little tricky

```
STAT #8 id=1 cnt=348 pid=0 pos=1 obj=0 op='SORT GROUP BY (cr=494 pr=19 pw=0
time=603190 us)'
```

```
STAT #8 id=2 cnt=2399 pid=1 pos=1 obj=0 op='HASH JOIN (cr=494 pr=19 pw=0
time=595897 us)'
```

```
STAT #8 id=3 cnt=437 pid=2 pos=1 obj=57036 op='TABLE ACCESS FULL ORD2
(cr=184 pr=11 pw=0 time=24947 us)'
```

```
STAT #8 id=4 cnt=70975 pid=2 pos=2 obj=57039 op='TABLE ACCESS FULL ORD_ITEM2
(cr=310 pr=8 pw=0 time=212985 us)'
```

- The values at the start of the line are for that line only. For example, the **cnt** of 2,399 on the second line means that step touched 2,399 rows.
- The values at the end of the line, within the parentheses are cumulative values. For example, the **cr** value of 494 on the second step includes the two steps below it because they are children of the **HASH JOIN** step.

## The CR value

---

Use the CR value to compare different versions of SQL

```
select count(*) from ord3
where rtn_date_convert(gmt_ord_date) > sysdate - 480 ;
```

Because of the function on the indexed column a full table scan is done.

```
STAT #13 id=1 cnt=1 pid=0 pos=1 obj=0 op='SORT AGGREGATE (cr=156 pr=0 pw=0
time=96834 us) '
STAT #13 id=2 cnt=0 pid=1 pos=1 obj=55663 op='TABLE ACCESS FULL ORD3 (cr=156
pr=0 pw=0 time=96795 us) '
```

Using a function based index allows this query to do much less work.

```
STAT #16 id=1 cnt=1 pid=0 pos=1 obj=0 op='SORT AGGREGATE (cr=2 pr=0 pw=0
time=79 us) '
STAT #16 id=2 cnt=0 pid=1 pos=1 obj=55668 op='INDEX RANGE SCAN GMT_ORDDT_IDX
(cr=2 pr=0 pw=0 time=47 us) '
```

## The TIME values

---

The TIME values is especially helpful when attacking an individual SQL statement

```
STAT #8 id=1 cnt=348 pid=0 pos=1 obj=0 op='SORT GROUP BY (cr=494 pr=19 pw=0
time=603190 us) '
STAT #8 id=2 cnt=2399 pid=1 pos=1 obj=0 op='HASH JOIN (cr=494 pr=19 pw=0
time=595897 us) '
STAT #8 id=3 cnt=437 pid=2 pos=1 obj=57036 op='TABLE ACCESS FULL ORD2
(cr=184 pr=11 pw=0 time=24947 us) '
STAT #8 id=4 cnt=70975 pid=2 pos=2 obj=57039 op='TABLE ACCESS FULL ORD_ITEM2
(cr=310 pr=8 pw=0 time=212985 us) '
```

- Each parent step includes all the children
- Need to subtract out the children's times to find out how long the parent really took. For example:
  - HASH JOIN time for self:  $595,897 - (24,947 + 212,985) = 357,965$
  - SORT GROUP BY time for self:  $60,3190 - 59,5897 = 7,293$
- Therefore the HASH JOIN is taking most of the time in this plan.

## 11g adds in more information (11.1.0.6).

```
STAT #2 id=1 cnt=13 pid=0 pos=1 obj=0 op='MERGE JOIN (cr=6 pr=0 pw=0
time=178 us cost=5 size=390 card=13)'
```

```
STAT #2 id=2 cnt=4 pid=1 pos=1 obj=71267 op='TABLE ACCESS BY INDEX ROWID
DEPT (cr=4 pr=0 pw=0 time=39 us cost=2 size=52 card=4)'
```

```
STAT #2 id=3 cnt=4 pid=2 pos=1 obj=71268 op='INDEX FULL SCAN DEPT_DEPTNO_PK
(cr=2 pr=0 pw=0 time=17 us cost=1 size=0 card=4)'
```

```
STAT #2 id=4 cnt=13 pid=1 pos=2 obj=0 op='SORT JOIN (cr=2 pr=0 pw=0 time=38
us cost=3 size=221 card=13)'
```

```
STAT #2 id=5 cnt=13 pid=4 pos=1 obj=71269 op='TABLE ACCESS BY INDEX ROWID
EMP (cr=2 pr=0 pw=0 time=110 us cost=2 size=221 card=13)'
```

```
STAT #2 id=6 cnt=13 pid=5 pos=1 obj=71274 op='INDEX FULL SCAN EMP_DEPT_IDX
(cr=1 pr=0 pw=0 time=34 us cost=1 size=0 card=13)'
```

- Cost – The Cost Column in V\$SQL\_PLAN
- Card – The Cardinality Column in V\$SQL\_PLAN
- Size – The Bytes column in v\$SQL\_PLAN

## Trace data (10046) and timings in 11g



- Timings in 11g stat lines from a 10046 trace file don't add up correctly when there is a sort in the plan
- A few bugs in Metalink that seem related to this:
  - 9090852: incorrect time information in 10046 trace (11.1.0.7)
  - 7522002: psrc: timing flaw in 10046 sql trace data (11.1.0.7)
  - 7168259: tst&perf:incorrect elapsed\_time in v\$sql and v\$sqlstats (11.2)
  - 8503195: wrong row source time statistics under sql\_trace=true (11.2)



## Trace data and timings in 11.1.0.6

```
SELECT * FROM EMP E, DEPT D WHERE E.DEPTNO = D.DEPTNO
/
```

```
MERGE JOIN ( time=144 us) SELF= 78?
TABLE ACCESS BY INDEX ROWID DEPT ( time= 44 us) SELF= 18
INDEX FULL SCAN DEPT_DEPTNO_PK ( time= 26 us) SELF= 24
SORT JOIN ( time= 22 us) SELF= -50
TABLE ACCESS BY INDEX ROWID EMP ( time= 72 us) SELF= 48
INDEX FULL SCAN EMP_DEPT_IDX ( time= 24 us) SELF= 24
```

- Parent time includes children
- Sort Join step took less then the children

Note: Stat lines were reformatted for easier reading.

## It gets worse, 11.1.0.7



```
SELECT * FROM EMP E, DEPT D WHERE E.DEPTNO = D.DEPTNO
/
```

```
MERGE JOIN ( time= 0 us) SELF= ?
TABLE ACCESS BY INDEX ROWID DEPT ( time= 0 us) SELF= ?
INDEX FULL SCAN DEPT_DEPTNO_PK ( time= 0 us) SELF= ?
SORT JOIN ( time= 0 us) SELF= ?
TABLE ACCESS BY INDEX ROWID EMP ( time= 0 us) SELF= ?
INDEX FULL SCAN EMP_DEPT_IDX ( time= 0 us) SELF= ?
```

- The time stats are **zero**
- This seems to affect the c and e times as well (the DB calls)

Note: Stat lines were reformatted for easier reading.

It gets better, seems fixed in 11.2.0.2



```
SELECT * FROM EMP E, DEPT D WHERE E.DEPTNO = D.DEPTNO  
/
```

```
MERGE JOIN (time= 186241 us) SELF= 37  
TABLE ACCESS BY INDEX ROWID DEPT (time= 84294 us) SELF= 58036  
INDEX FULL SCAN DEPT_DEPTNO_PK (time= 26258 us) SELF= 26258  
SORT JOIN (time= 101910 us) SELF= 122  
TABLE ACCESS BY INDEX ROWID EMP (time= 101788 us) SELF= 69622  
INDEX FULL SCAN EMP_DEPT_IDX (time= 32166 us) SELF= 32166
```

Note: Stat lines were reformatted for easier reading.

React



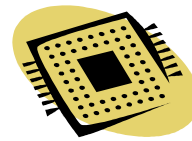
## Some Common Wait events and what to do

---

- Once you trace someone's session, you will need to create a profile of the events at depth 0.
- Order this profile based on which will show which event consumed the most time.
- Depending on which event is at the top (the one that consumed the most time) you will need to take an appropriate action.
- This is a very brief overview of the some common events and the basic actions to respond to these events.
- There are 1,140 wait events in 11.2.0.2, but you will likely only deal with a small set of these.

**CPU service**  
**CPU service, latch free**

---



How do you respond when **CPU service** is the dominant contributor in your profile?

---

1. Identify the responsible high-**c** db calls
2. If high **c** value is because of recursive children, then go to 1
3. If high **c** value for **FETCH** or **EXEC**
  - LIO problem? Then tune SQL
  - Sort operations? Then fix SQL, or use indexes to avoid sorting
  - PL/SQL processing? Then **DBMS\_PROFILER**
4. If high **c** value for **PARSE**
  - Unnecessary parse call? Then eliminate it
  - Otherwise, reduce the work required to do the parse

db file scattered read,  
db file sequential read

---



How do you respond when `db file % read` is the dominant contributor in your profile?

---

1. If it is dominant because of a few latencies?
  - a) Eliminate the program's unnecessary I/Os to the device
  - b) Eliminate unnecessary I/O competition to this device
    - Fix other actions' SQL
    - Move low-priority jobs to other times
    - Upgrade your disks or disk architecture (e.g., abandon RAID 5, which penalizes readers because small-write operations consume so much I/O capacity)
2. If it is dominant because of too many reads?
  - a) Then fix your SQL
  - b) If SQL is optimized, then consider a bigger buffer cache

`SQL*Net message from client`

---



How do you respond when **SQL\*Net message from client** is the dominant contributor in your profile?

---

1. Confirm that duration is actually part of someone's *R* (*response time*)
2. Dominant because of high #calls? Then why so many distinct db calls?
3. Dominant because of a few latencies? Then why does app spend so much time between db calls?
4. Can't reduce #calls? Then reduce other programs' unnecessary use of your network capacity

**buffer busy waits**

---



How do you respond when **buffer busy waits** is the dominant contributor in your profile?

---

1. Identify the block being contended for (**p1=file#, p2=block#**) and the reason for the contention (**p3=id**); see MetaLink 34405.1
  - Poor batch partitioning? Then partition your input stream so that concurrent jobs don't compete for the same buffers
  - Poor application design? Then design the application so that concurrent users don't serialize on a small number of buffers
  - Poor data storage? Then store data more sparsely to relieve competition
  - Poor SQL? Then fix the SQL to eliminate unnecessary LIOs that compete for buffers

---

Q  
QUESTIONS  
&  
ANSWERS