Diagnosing the past using AWR: hard parse headache

The **A**utomatic **W**orkload **R**epository report is an excellent tool for diagnosing and troubleshooting past performance issues. It continuously gathers valuable performance key indicators without any known noticeable side effects. Unfortunately it comes with two important drawbacks (a) it requires an additional license fees and (b) it needs a solid technical Oracle background including the right combination of rules with some sort of context, time-frame, and information cross check. The free AWR alter-ego Statspack can be used to overcome the extra license drawback. As for the solid background required for the correct interpretation, this list of reliable references about reading and interpreting AWR and Statspack reports collected by Jonathan Lewis represents a good starting point. Nicolay Savinov did also a nice work in popularising AWR report interpretation via real life practical cases. Franck Pachot presentation Interpreting AWR reports – straight to the Goal represents, as well, an elegant and practical first approach towards reading and interpreting AWR reports. The first part of this chapter takes you through an artificially created performance issue triggered by an excessive hard parse anomaly and, over which, an AWR snapshot will be taken. The second section starts by presenting the main sections of the generated AWR report from which important performance key indicators will be emphasized. It then shows how to locate, read, and interpret the symptoms of the provoked hard parse issue in the different sections of the AWR report and how to correlate them properly. Section 3 moves onto suggestions of solving the hard parse issue, addresses it and shows how the previously identified critical section of the AWR report has changed after the fix.

1. Setting the hard parse scene

While the majority of Oracle hard parse issues is primarily due to non-shared **parent cursors**, there are, nevertheless, situations where a hard parse activity can be exacerbated by non sharing **child cursors**. This article deals only with the first kind of cursors. Excessive hard parse due to non-sharing child cursor might probably be a subject of a separate article.

In order to provoke a hard parse storm we will use Tanel Poder **lotshparses.sql** script. This very simple PL/SQL anonymous block looks like the following:

```
declare
   x number;
begin
   for i in 1..&1
   loop
     execute immediate 'select count(*) from dual where rownum = '||
                                  to_char(dbms_random.random) into x;
   end loop;
end;
/
```

As you can see this script executes several times the following query:

```
SQL> select count(*) from dual where rownum = random_hard_coded_value;
```

Since the pseudo column *rownum* is compared to a random hard coded value, for each execution of the above query, Oracle will create a new parent cursor. This will inevitably fill up the SGA with a bunch of different **SQL_ID** having the same force matching signature (see later in this article). Furthermore, this kind of non-sharing parent cursors issue is not easily diagnosable in an AWR report because the workload is spread over several different, and eventually rapid, cursors having the same semantically equivalent SQL code.

In order to cover the hard parse storm in its totality we are going to create an AWR snapshot just before launching the **lotshparses.sql** script and immediately after its end as shown below:

```
SQL> set timing on

SQL> exec dbms_workload_repository.create_snapshot;

SQL> @lotshparses 200000

PL/SQL procedure successfully completed.

Elapsed: 00:04:25.48

SQL> exec dbms_workload_repository.create_snapshot;
```

Proceeding as such we will have an AWR report perfectly covering the hard parse activity duration.

2. Generating the AWR report

One of the simplest ways I use for generating an AWR report is executing Tanel Poder **gen_awr_report** script directly from a SQL Plus session as shown below:

```
SQL> @gen_awr_report
Listing latest AWR snapshots ...

   SNAP_ID END_INTERVAL_TIME
---------- ------------------------------
        64 17/08/17 20:57:07,197
        65 18/08/17 03:12:48,044
        66 18/08/17 07:49:16,415
        67 18/08/17 07:51:05,681
        68 18/08/17 07:51:51,661
        69 18/08/17 07:53:25,703
        70 18/08/17 07:53:53,423
        71 18/08/17 07:54:55,435
        72 18/08/17 07:57:28,508
        73 18/08/17 07:58:18,818
        74 18/08/17 14:16:34,616
        75 18/08/17 18:31:36,362
        76 18/08/17 19:00:11,736
        77 18/08/17 19:06:02,127

14 rows selected.

Enter begin snapshot id: 76
Enter   end snapshot id: 77

PL/SQL procedure successfully completed.
```

The user launching the gen_awr_report script needs, however, to have been granted the following privilege:

```
SQL> grant execute on dbms_workload_repository to user;
```

The begin and end snapshot id should correspond to the snapshot we have created before the start and after the end of the **lotshparses** script. The html AWR report will be generated in the directory from which the **gen_awr_report** script has been launched.

3. Diagnosing excessive hard parse in AWR

3.1. DB time load

The first section of the AWR reports shows the start and the end time of the snapshot together with the Elapsed and DB time:

	Snap Id	Snap Time	At Sessions	Cursors/Session	Pluggable Databases Open
Begin Snap:	76	18-Aug-17 19:00:11	38	1.0	0
End Snap:	77	18-Aug-17 19:06:02	41	1.2	0
Elapsed:		5.84 (mins)			
DB Time:		4.45 (mins)			

Let's detail a little bit the above information:

- Elapsed time : 5.84 mins : represents the delta time between the two AWR snapshots (76 and 77)
- DB time : 4.45 mins : represents the time spent by all active sessions during the Elapsed time

Typically real life running systems gather data at one hour intervals and keep 45 days of historical retention. What is important to remember is that, a helpful and exploitable AWR report, should have a snapshot covering the period in which the performance pain occurred. In the current case, we have managed to enclose the execution of the **lotshparses** script between two distinct AWR snapshots such that we will have an AWR report covering perfectly the script time duration period.

DB time is the sum of time spent by all foreground sessions in the database. The more DB time is greater than Elapsed time the more likely the application will be suffering a performance pain. Expressed differently this can be turned to: the more your database is overloaded the more it will suffer a performance problem. This is why it is extremely important to figure out the database load as early as possible when interpreting an AWR report.

A **database load**, also known as the **average active session**, can be computed using the following simple formula:

```
DB time/Elapsed time = 4.45/5.84 = 0.76
```

The above calculated value signifies that during the 5,84 minutes of elapsed wall clock time there was, in average, constantly 0.76 active sessions. Any connected session can consume 5.84 minutes. This is why when we divide DB time by the Elapsed time we expect to have the number of sessions that were active during the AWR snapshot. An important point which is worth a reminder is that by active session we are referring to:

- sessions actively burning CPU
- sessions actively waiting for a non-idle Oracle instrumented I/O wait event or lock

Simply put, figuring out the number of average active session is the first performance key indicator to clear out when interpreting an AWR report either for a global performance issue or for a pro-active audit. For example, suppose that you are supplied with an AWR report having the following workload:

	Snap Id	Snap Time	At Sessions	Cursors/Session
Begin Snap:	26011	28-Feb-16 08:00:26	65	2
End Snap:	26012	28-Feb-16 09:00:31	64	2
Elapsed:		60.09 (mins)		
DB Time:		5.46(mins)		

It is clear that this database during this AWR snapshot period was almost idle since its database workload is very small (5.46/60 = 0.09).

However an AWR report having the following database workload (3555/60 = 59) is worth a deep investigation:

	Snap Id	Snap Time	At Sessions	Cursors/Session
Begin Snap:	26011	28-Feb-16 08:00:26	65	2

End Snap:	26012	28-Feb-16 09:00:31	64	2
Elapsed:		60.09 (mins)		
DB Time:		3555(mins)		

3.2. DB CPU load

We can also derive the CPU Load in the same way as we did to calculate the DB time load. This can be simply achieved by dividing the DB CPU time by the number of available CPU cores:

Host Name	Platform	CPUs	Cores	Sockets	Memory (GB)
HP-PC	Microsoft Windows x86 64-bit	4	2	1	15.87

```
DB CPU Load = DB CPU time/Elapsed time = 266,3/(5.84*60) = 0.7599
```

The DB CPU time value can generally be found either into the **Top 10 Foreground Events by Total Wait Time** section of the AWR report (see below) or within the **Time Model Statistics** section f the same report.

However it is important to know that the DB CPU load we have derived above is for a single CPU machine. In order to have the correct CPU load for the current application we have to divide the above DB CPU load by the number of available cores in this machine:

```
DB CPU Load = DB CPU load/2 = 0.7599/2= 37,9%
```

Finally we can say that, during the hard parse storm we have provoked, we were almost burning 38% of our total available CPU.

3.1. Load Profile

The Load Profile, although very often neglected, is a critical section of the AWR report as it gathers fundamental information about the global health check of the application. The Load Profile of the current situation is shown below:

Load Profile

	Per Second	Per Transaction	Per Exec	Per Call
DB Time(s):	0.8	17.8	0.00	1.96
DB CPU(s):	0.8	17.8	0.00	1.96
Background CPU(s):	0.0	0.1	0.00	0.00
Redo size (bytes):	7,405.6	172,991.2		
Logical read (blocks):	138.0	3,223.5		
Block changes:	29.3	683.3		
Physical read (blocks):	0.3	7.3		
Physical write (blocks):	2.0	46.7		
Read IO requests:	0.3	7.3		
Write IO requests:	1.0	23.3		
Read IO (MB):	0.0	0.1		
Write IO (MB):	0.0	0.4		
IM scan rows:	0.0	0.0		
Session Logical Read IM:	0.0	0.0		
User calls:	0.4	9.1		
Parses (SQL):	574.5	13,420.2		
Hard parses (SQL):	571.3	13,345.7		
SQL Work Area (MB):	0.7	16.7		
Logons:	0.1	1.9		
Executes (SQL):	607.3	14,186.3		

Rollbacks:	0.0	0.0
Transactions:	0.0	

As you can see the Load Profile section starts by displaying two fundamental values

	Per Second	Per Transaction	Per Exec	Per Call
DB Time(s):	0.8	17.8	0.00	1.96
DB CPU(s):	0.8	17.8	0.00	1.96

If you are wondering what these DB time and DB CPU information are then bear in mind that they are just rounding to the single decimal digit of the DB time load and DB CPU load we have derived above as the followings prove:

```
DB Time(s):  = round(DB time load,1) = round (0.76,1)   = 0.8
DB CPU(s):   = round(DB CPU load,1)  = round (0.7599,1) = 0.8
```

As per regards to the parsing information which I have managed to red bold its value in the load profile displayed above, the application did 574 parses per second of which 571 are hard parses. That is to say the least, more than 99% of application parses resulted into hard parses. That's the first observation about the application parsing activity we can infer from this AWR report. Note by the way that despite this high rate of hard parsing activity we are still unable to say whether this represents a problem or not for the whole application.

Let's move on and see what parsing information we can found in the Instance Efficiency section.

3.2. Instance Efficiency Percentages (Target 100%)

The high hard parse ratio pointed out in the above Load Profile section is confirmed a little bit down the same AWR report at the instance efficiency section as the following demonstrates:

Buffer Nowait %:	100.00	Redo NoWait %:	100.00
Buffer Hit %:	99.77	In-memory Sort %:	100.00
Library Hit %:	55.91	Soft Parse %:	0.56
Execute to Parse %:	5.40	Latch Hit %:	99.99
Parse CPU to Parse Elapsd %:	99.68	% Non-Parse CPU:	01/12/80
Flash Cache Hit %:	0.00		

The 5,40 *Execute to Parse* ratio is another indication of the excessive hard parsing rate done by the application:

```
Execute to Parse% =  1 - Parses/Executes = 1 - (574,5/607,3) = 5,4%
```

Notice in passing that while the system seems to be quite busy hard parsing it did nevertheless a minimum of soft parses as indicated by the tiny 0.56% value of the *Soft Parse* ratio. This value is computed using the following formula (values come from the Load Profile):

```
Soft Parse% = (Parses - Hard parses)/Parses = (574.5-571.3)/574.5=0.56%
```

The 99,68% ***Parse CPU to Parse Elaspd%*** value indicates that the CPU time used by this application for parsing represents 99,68% of the total elapsed parse time. Expressed differently this means that for every available CPU second the application spent about 100/99,68 = 1,003 second wall clock time parsing. The 99,68% ratio is calculated using the Instance Activity Stats section shown below:

3.3. Instance Activity Stats

Statistic	Total	per Second	per Trans

opened cursors cumulative	212,687	607.00	14,179.13
parse count (describe)	0	0.00	0.00
parse count (failures)	4	0.01	0.27
parse count (hard)	200,185	571.32	13,345.67
parse count (total)	201,303	574.51	13,420.20
parse time cpu	23,22	66.28	1,548.27
parse time elapsed	23,3	66.49	1,553.27

```
Parse CPU to Parse Elapsd % = parse time cpu /parse time elapsed

                              = 23,22/23,3=99,65%
```

There is one important point to emphasize here. The Instance Efficiency section states that 100% is the target and ideal value. Do then the 99,68% of Parse CPU to Parse Elapsd% metric imply that the current system is not burning a lot of CPU during parsing? In fact this is a perfect illustration of how an AWR information can be misleading and self-contradictory when it is considered in isolation without being cross-checked with other metrics from different sections of the same report. The 99,68% of Parse CPU to Parse Elapsd% value is simply indicating that, roughly, all parsing time has has been spent burning the server's CPU. In itself this seems to be good news as we don't want our parsing activity to be waiting for any I/O contention. However, the absence of I/O contention during a parsing activity doesn't necessarily mean that there is nothing to worry about. Practical real life cases tend to suggest that, ideally, we would not like to see the parsing activity burning a large percentage of the DB CPU time. We would rather expect to see that most of the CPU time be consumed running user SQL statements. But let's note at this stage of the investigation that our application was not waiting on any I/O during the parse activity and let's continue our investigations by looking, this time, to the Instance Activity Stats section.

3.4 Key Instance Activity Stats

The total parsing activity is reported in the key instance activity stats of the AWR report reproduced here below:

Statistic	Total	per Second	per Trans
db block changes	10,249	29.25	683.27
execute count	212,795	607.31	14,186.33
logons cumulative	29	0.08	1.93
opened cursors cumulative	212,687	607.00	14,179.13
parse count (total)	201,303	574.51	13,420.20
parse time elapsed	23,299	66.49	1,553.27
physical reads	109	0.31	7.27
physical writes	701	2.00	46.73
redo size	2,594,868	7,405.64	172,991.20
session cursor cache hits	12,045	34.38	803.00
session logical reads	48,352	137.99	3,223.47
user calls	136	0.39	9.07
user commits	15	0.04	1.00
user rollbacks	0	0.00	0.00
workarea executions - optimal	440	1.26	29.33

```
execute count / parse count (total) = 212,795/201,303 = 1,057
```

As it has already been pointed out in the Load Profile section, there is roughly one hard parse per execution where ideally there should be one parse for many executions.

3.5 Time Model Statistics

Although all the above cross checked information tends to suggest the presence of an exaggerated hard parse activity we still have no idea of how significant its impact might be on the overall response time of the application. In order to have an idea of how much benefit we will get by addressing the hard parse activity, we need first to evaluate the percentage of DB time against the parse time. This is achievable via the information supplied by The ***Time Model Statistics*** section of the AWR report shown below:

Statistic Name	Time (s)	% of DB Time	% of Total CPU Time
sql execute elapsed time	267.10	99.96	
DB CPU	266.32	99.67	99.61
parse time elapsed	232.87	87.15	
hard parse elapsed time	190.05	71.13	
PL/SQL execution elapsed time	13.39	5.01	
connection management call elapsed time	0.05	0.02	
PL/SQL compilation elapsed time	0.02	0.01	
repeated bind elapsed time	0.00	0.00	
failed parse elapsed time	0.00	0.00	
hard parse (sharing criteria) elapsed time	0.00	0.00	
sequence load elapsed time	0.00	0.00	
DB time	267.20		
background elapsed time	17.63		
background cpu time	1.05		0.39
total CPU time	267.37		

The application consumed 267 seconds of DB time of which 190 seconds have been spent hard parsing. Addressing the hard parse issue is thereby going to probably enhance the application response time by a factor of 71% (190/267). Don't be surprised if you have already noticed that the % of DB time column above doesn't sum up to 100%. This is because the DB CPU time metric is part of the DB time as we need CPU to parse and execute sql statements.

The next section of the AWR report has been precisely implemented to help us identifying how the top 10 foreground events are distributed by total wait time as shown below:

3.6 Top 10 Foreground Events by Total Wait Time

Event	Waits	Total Wait Time (sec)	Avg Wait	% DB time	Wait Class
DB CPU		266,3		99.7	
control file sequential read	164	0	218.13us	.0	System I/O
latch: shared pool	55	0	171.18us	.0	Concurrency
db file sequential read	24	0	271.21us	.0	User I/O
SQL*Net more data from client	16	0	390.19us	.0	Network
PGA memory operation	436	0	9.85us	.0	Other
Disk file operations I/O	5	0	530.60us	.0	User I/O
SQL*Net message to client	52	0	1.98us	.0	Network
log file sync	1	0	33.00us	.0	Commit
row cache mutex	2	0	5.00us	.0	Concurrency

As you can easily see almost all the elapsed time has been spent burning CPU (263 seconds) compared to the next wait time. However, as already explained above high CPU utilization may not necessarily mean that there is a problem; it could just mean that the system is being well utilized and that the CPU time is used to run SQL statements. If we want to know where this CPU time has been burned we need to drill down to the section of the AWR report that collects TOP SQL statements by CPU time utilization reproduced here below:

3.7 SQL ordered by CPU Time

CPU Time (s)	Executions	CPU per Exec (s)	%Total	Elapsed Time (s)	%CPU	%IO	SQL Id	SQL Module	SQL Text
265.25	1	265.25	99.60	265.48	99.91	0.00	6ht3qvxf0gwxu	SQL*Plus	declare x number; begin for i ...
0.84	1	0.84	0.32	0.93	90.44	1.41	cbg8aua031a9g	SQL*Plus	dbms_workload_reposit
0.45	10,629	0.00	0.17	0.34	135.00	0.00	87gaftwrm2h68		select o.owner#, o.name, o.nam...

Finally we ended up by finding that the PL/SQL anonymous block(`6ht3qvxf0gwxu`) of the **lotshparses.sql** script is the piece of code responsible of 99% of the 266 seconds spent burning CPU.

And, as already explained above, because we didn't use bind variables values when executing the **lotshparses.sql** script, Oracle has distributed the corresponding workload over a bunch of different parent SQL_ID rapidly executed so that they have not been captured into the AWR historical tables.

4. Diagnosing excessive hard parse at real time basis

For the sake of completeness and comprehensive concept, let's create a breach in this past performance diagnostic methodology, by analysing the provoked hard parse storm at a real time basis. While the **lotshparses.sql** script was still running I managed to take several snapper(Tanel Poder script) snapshots to see what was precisely happening behind the scenes:

```
SQL> @snapper ash 5 1 all
Sampling SID all with interval 5 seconds, taking 1 snapshots...

------------------------------  ----------------------------------
Active% | INST | SQL_ID          | SQL_CHILD | EVENT    | WAIT_CLASS
------------------------------------------------------------------
     9% |    1 | 6ht3qvxf0gwxu   | 0         | ON CPU | ON CPU
     4% |    1 |                 | 0         | ON CPU | ON CPU
     2% |    1 | 66g7vv2ms73au   | 0         | ON CPU | ON CPU
     2% |    1 | 9cwfbxxsnsspt   | 0         | ON CPU | ON CPU
     2% |    1 | cv00j9jn17pdj   |           | ON CPU | ON CPU
     2% |    1 | 7dwajvra4y7nd   | 0         | ON CPU | ON CPU
     2% |    1 | 0tv4xawx6wza5   | 0         | ON CPU | ON CPU
     2% |    1 |                 |           | ON CPU | ON CPU
     2% |    1 | f19xaxh25spmc   |           | ON CPU | ON CPU
     2% |    1 | 6x9ss1wkjqms7   | 0         | ON CPU | ON CPU

--  End of ASH snap 1, end=2017-08-18 19:03:53, seconds=5, samples_taken=47

-------------------------------------------------------------------
Active% | INST | SQL_ID          | SQL_CHILD | EVENT    | WAIT_CLASS
-------------------------------------------------------------------
     9% |    1 | 6ht3qvxf0gwxu   | 0         | ON CPU   | ON CPU
     4% |    1 |                 | 0         | ON CPU   | ON CPU
     2% |    1 | 893u29ac8qsm7   | 0         | ON CPU   | ON CPU
     2% |    1 | a1kk4h1k4fu6k   | 0         | ON CPU   | ON CPU
     2% |    1 | 7p397shk05hdk   | 0         | ON CPU   | ON CPU
     2% |    1 | fd5nuwp4wf0ty   | 0         | ON CPU   | ON CPU
     2% |    1 | babdqtsna9qhv   | 0         | ON CPU   | ON CPU
     2% |    1 | 32wf8npszsg35   | 0         | ON CPU   | ON CPU
     2% |    1 | ctu5x7p45vf37   | 0         | ON CPU   | ON CPU
     2% |    1 | atkkb0bu1g6gs   | 0         | ON CPU   | ON CPU

--  End of ASH snap 1, end=2017-08-18 19:04:02, seconds=5, samples_taken=46

---------------------------------------------------------------------
Active% | INST | SQL_ID          | SQL_CHILD | EVENT      | WAIT_CLASS
---------------------------------------------------------------------
```

```
     7% |     1 | 6ht3qvxf0gwxu    | 0          | ON CPU     | ON CPU
     5% |     1 |                  |            | ON CPU     | ON CPU
     5% |     1 |                  | 0          | ON CPU     | ON CPU
     2% |     1 | 9pc6rdmpw9avy    | 0          | ON CPU     | ON CPU
     2% |     1 | fub0xjvsmgryp    | 0          | ON CPU     | ON CPU
     2% |     1 | d8u6459pc9y55    | 0          | ON CPU     | ON CPU
     2% |     1 | 72hypdu5ggwur    |            | ON CPU     | ON CPU
     2% |     1 | ct8whvpph16bm    | 0          | ON CPU     | ON CPU
     2% |     1 | cktrahgwngpz0    | 0          | ON CPU     | ON CPU
     2% |     1 | 16qs7uvs32h6j    | 0          | ON CPU     | ON CPU

--  End of ASH snap 1, end=2017-08-18 19:04:10, seconds=5, samples_taken=43
```

As you can see there are a bunch of new different sql_id appearing at each snapper snapshot. But there is also a repeated SQL_ID (`6ht3qvxf0gwxu`) on the top of each snapshot. It is easily guessable that the repeated SQL_ID refers to the anonymous PL/SQL block while the brand new SQL_IDs refer to an instance of the same sql statement using different hard coded value.

If we look closely at those semantically equivalent sql statements, we will find that, indeed, they share the same force matching signature as the following shows:

```
SQL> select
       to_char(force_matching_signature), count(1)
     from
        gv$sql
     where
        sql_text like '%select count(*) from dual%'
     and sql_text not like '%v$sql%'
     group by
      to_char(force_matching_signature)
     order by 2 desc;

TO_CHAR(FORCE_MATCHING_SIGNATURE)          COUNT(1)
---------------------------------------- ----------
3674816115711089083                            3427
12435787424289997025                           3414
0                                                 1
```

The sql_text behind the first two force matching signature are respectively:

```
SQL> select
        sql_text
     from
        gv$sql
     where
       force_matching_signature = '3674816115711089083';

select count(*) from dual where rownum = 577153088
select count(*) from dual where rownum = 1839045909
select count(*) from dual where rownum = 286104478
select count(*) from dual where rownum = 94480864
select count(*) from dual where rownum = 1184838553

3427 rows selected.

SQL> select
       sql_text
     from
       gv$sql
     where
      force_matching_signature = '12435787424289997025';

select count(*) from dual where rownum = -626593942
select count(*) from dual where rownum = -1998806196
select count(*) from dual where rownum = -1630012206
select count(*) from dual where rownum = -295277308
select count(*) from dual where rownum = -600932638

3414 rows selected.
```

Instead of being parsed once and executed many times, these multiple instances of the same query are never re-executed (executions = 1) as shown below via the following simple query:

```
SQL> select
          *
     from
        (select
            substr(sql_text,1,70) text
           ,executions
         from
             gv$sql
          where
             sql_text like '%select count(*) from dual%'
          and  sql_text not like '%gv$sql%'
          and  sql_text not like '%declare%'
          and  executions <=2
          );
```

```
TEXT                                                      EXECUTIONS
------------------------------------------------------ ----------
select count(*) from dual where rownum = 939734968              1
select count(*) from dual where rownum = -1006924429            1
select count(*) from dual where rownum = -114275733             1
select count(*) from dual where rownum = 1953948273             1
select count(*) from dual where rownum = 1124821383             1
select count(*) from dual where rownum = 610219080              1
select count(*) from dual where rownum = -1946236830            1
select count(*) from dual where rownum = 1287820255             1
select count(*) from dual where rownum = -999606527             1
select count(*) from dual where rownum = 1255763693             1
select count(*) from dual where rownum = 1937225088             1
select count(*) from dual where rownum = 682402740              1
select count(*) from dual where rownum = 50301004               1
select count(*) from dual where rownum = -385453719             1
select count(*) from dual where rownum = -977713588             1
```

../..

In the next section we will show how we can fix this parsing issue.

5. Addressing the hard parse issue

In order to work around the above hard parse issue we can force the value of the cursor_sharing parameter so that Oracle will use system bind variables behind the scenes making the above instances of the same query sharing the same parent cursor (SQL_ID):

```
SQL> alter session set cursor_sharing=force;
```

Once this parameter changed the following will re-execute the same script enclosed between two different AWR snapshots as shown below:

```
SQL> exec dbms_workload_repository.create_snapshot;

SQL> @lotshparses 200000

PL/SQL procedure successfully completed.

Elapsed: 00:00:19.40
```

```
SQL> exec dbms_workload_repository.create_snapshot;

PL/SQL procedure successfully completed.
```

As you can see the script completed, this time, very quickly than before as it necessitated 19 seconds instead of the previous 4 minutes.

Before examining the new hard parse metrics gathered into the new AWR report let's see now how attractive is the shared memory (library cache) following the cursor sharing parameter change:

```
SQL> select
         *
     from
        (select
             substr(sql_text,1,70) text
            ,executions
         from
            gv$sql
         where
             sql_text like '%select count(*) from dual%'
         and  sql_text like '%SYS_B%'
         and  sql_text not like '%gv$sql%'
         and  sql_text not like '%declare%'
         );
```

```
 TEXT                                                   EXECUTIONS
------------------------------------------------------ ----------
select count(*) from dual where rownum = :"SYS_B_0"        100009
select count(*) from dual where rownum = -:"SYS_B_0"        99989
```

Spot how Oracle has replaced the literal values by system variables (`SYS_B_0`) and ended up by re-executing he same query hundreds of thousands of times:

The **Load Profile** of the new AWR report backs up the above new attractive parsing situation as shown below (limited only to information related to the parsing activity):

	Snap Id	Snap Time	Sessions	Cursors/Session	Pluggable Databases Open
Begin Snap:	2570	27-Oct-18 16:41:42	44	1.0	0
End Snap:	2571	27-Oct-18 16:42:04	44	1.0	0
Elapsed:		0.36 (mins)			
DB Time:		0.33 (mins)			

Load Profile

	Per Second	Per Transaction	Per Exec	Per Call
DB Time(s):	0.9	1.7	0.00	5.01
DB CPU(s):	0.9	1.7	0.00	5.00
Background CPU(s):	0.0	0.0	0.00	0.00
Redo size (bytes):	115,496.8	209,925.0		
Logical read (blocks):	781.6	1,420.7		
Block changes:	359.7	653.8		
Physical read (blocks):	0.1	0.1		
Physical write (blocks):	0.1	0.1		
Read IO requests:	0.1	0.1		
Write IO requests:	0.1	0.1		
Read IO (MB):	0.0	0.0		

Write IO (MB):	0.0	0.0
IM scan rows:	0.0	0.0
Session Logical Read IM:	0.0	0.0
User calls:	0.2	0.3
Parses (SQL):	9,191.9	16,707.0
Hard parses (SQL):	0.6	1.0
SQL Work Area (MB):	2.0	3.5
Logons:	0.0	0.0
Executes (SQL):	9,233.8	16,783.3
Rollbacks:	0.0	0.0
Transactions:	0.6	

The script did 9,191 parses of which, as of now, only one hard parse. This is an impressive reduction of the hard parse activity thanks to the forced value of the cursor sharing parameter. A very low rate of hard parse activity represents an enormous boost to the scalability of the application.

The ***Instance Efficiency*** section of the same report shows as well a drastic improvement of the hard parse figures as shown below:

Instance Efficiency Percentages (Target 100%)

Buffer Nowait %:	100.00	Redo NoWait %:	100.00
Buffer Hit %:	99.99	In-memory Sort %:	100.00
Library Hit %:	99.97	Soft Parse %:	99.99
Execute to Parse %:	0.45	Latch Hit %:	100.00
Parse CPU to Parse Elapsd %:	95.25	% Non-Parse CPU:	72.95
Flash Cache Hit %:	0.00		

We went from an *Execute to Parse ratio* of 5,40 to a ratio of 0.45. The 99.99% of Soft Parse is another indication that the system is almost not hard parsing at all. Whether this very high rate of soft parsing is good or not is beyond the scope of this article.

Finally the new ***Time Model Statistics*** section presented below demonstrates that, from now on, the hard parse elapsed time (0.01 second) doesn't contribute anymore to the total amount of the DB time:

Statistic Name	Time (s)	% of DB Time	% of Total CPU Time
sql execute elapsed time	20.05	99.99	
DB CPU	20.00	99.74	100.00
parse time elapsed	6.31	31.45	
PL/SQL execution elapsed time	5.08	25.34	
hard parse elapsed time	0.01	0.06	
hard parse (sharing criteria) elapsed time	0.01	0.03	
PL/SQL compilation elapsed time	0.00	0.01	
repeated bind elapsed time	0.00	0.00	
DB time	20.05		
background elapsed time	1.29		
total CPU time	20.00		

6. Conclusion

Using an AWR report we can get the average load of a particular system. This report gives few pointers to where we might look and address in order to solve performance issues and reduce the overall workload of the system. Provided we know how to read it, cross check its information and interpret its critical parts, it can dramatically help us troubleshooting many performance issues. In this article, we examined how to diagnose a parent-cursor hard parse situation. We outlined that, in this context, ***Parses (SQL)*** and ***Hard parses (SQL)*** of the ***Load Profile*** section are the two critical metric to look at. The later should represent a big fraction of the former before to start thinking that your application is probably suffering from an excessive hard parse activity. We have also emphasized that, before trying to fix an excessive hard parsing problem, you should first evaluate the maximum of benefit you might get by addressing the hard parse activity. The ***Time Model Statistics*** section is then the area where you have to look in order to get a rough estimation of the plausible improvement. If the hard parse elapsed time represents a considerable fraction of the DB time then it is fairly likely that reducing the hard parse rate will improve the overall performance of the application. Finally we moved on to a simple work-around by forcing the value of the cursor sharing parameter and showed how this change affected positively the hard parse critical part of the new AWR report.