

# PERFORMANCE DIAGNOSTIC DATA AND STATSPACK: WHAT'S NEW IN ORACLE9I DATABASE RELEASE 2

*Graham Wood, Architect, Oracle Corporation*

## **INTRODUCTION**

The Oracle9i database provides a plethora of statistical information about its operation. This data can be used by DBAs to help them to analyze and diagnose the operation of their systems. This paper highlights some of the new statistics that have been introduced with the Oracle9i Database Release 1 (Oracle9iR1) and Oracle9i Database Release 2 (Oracle9iR2) and provides examples of how to make use of them. Have you ever wanted to know the answers to questions such as:

'What is the plan for the statement that has been executing for 35 minutes?'

'Does the new batch application make use of array operations?'

'How many IOs would be saved by increasing the buffer cache by 10%?'

One mechanism for reducing the volume of data that a DBA needs to look at is by using the Statspack tool that has shipped with the database since Oracle8i. Many of the new statistics are captured and reported either by default or optionally when using Statspack. We will discuss enhancements to Statspack made in Oracle9i Database releases.

The paper also highlights a few of the small but helpful enhancements that do not warrant marketing attention but can save a DBA and performance analysts more than a little time.

## NEW DIAGNOSTIC STATISTICS

Although the Oracle database provided many statistics, there are still issues that can be hard to diagnose. The purpose of adding yet more statistics is to help identify some of these issues and simplify diagnosis. New statistics are also added to allow the monitoring of new features and to replace old statistics that no longer offer valid information.

Examples of the latter are the statistics *sorts (disk)* and *sorts (memory)*. These statistics have long been used to determine what percentage of sorts were being performed in memory and what percentage were overflowing the sort area and being written to disk. Analysis of this data allowed the DBA tune the parameter *sort\_area\_size*. However as server software has changed and new operations, such as hash joins and bitmap operations have been introduced, these statistics no longer include all of the causes of writing workareas to disk. The new statistics introduced for *optimal, one pass* and *multipass workarea executions* are a much better measure of this activity.

One example of simplification is the new *statistics\_level* parameter. The intention of this parameter is to provide a single switch that allows the DBA to control the volume of data that will be maintained in the many fixed tables (v\$ tables). Releases prior to Oracle9iR2 required separate parameters to be set to enable specific data collections such as *timed statistics*, *db\_cache\_advice*, and *timed\_os\_statistics*. Setting *statistics\_level=TYPICAL* (the default) enables statistics capture at a level that will provide broad coverage while minimizing the performance overhead. Setting *statistics\_level=FULL* will enable more detailed, and also more intrusive, statistics to be maintained. The fixed table *v\$statistics\_level* gives details of which data is captured at the different levels.

### *SEGMENT STATISTICS (ORACLE9iR2)*

Segment statistics provide information about the level of usage of each segment in use in the database. These statistics allow the analysis of the operation of a system on an object-by-object basis to determine which objects are the subject of high resource utilization (for example, the most IOs or buffer busy waits). This data can be used in much the same way that *v\$sql* and *v\$filestat* can be used to identify high load SQL and hot files.

For each segment which has buffers in the buffer cache the following statistics are collected:

- buffer busy waits
- db block changes
- global cache current and CR blocks served (Real Application Clusters)
- logical reads
- physical reads and writes
- direct physical reads and writes
- ITL waits (share mode TX lock waits)
- row lock waits (exclusive mode TX lock waits)

The data is presented as a fixed table, *v\$segment\_statistics*, with one row for each segment for each of the above statistics. Entries appear in this table for all objects that have been *used* since instance startup rather than for all segments in the database. As an example of its use the SQL below shows the segments that have been the subject of most row lock waits since the database instance was started:

```
SQL> select object_name, statistic_name, value from v$segment_statistics
2  where statistic_name='row lock waits'
3  order by value desc;
```

OBJECT_NAME	STATISTIC_NAME	VALUE
OM_INSTANCE	row lock waits	225392
OM_HEADER	row lock waits	121369
ES_INSTANCE	row lock waits	73559

MWI_Q_TABLE	row lock waits	45038
ES_FOLDER	row lock waits	31865
ES_QUEUE	row lock waits	13734
ES_EXT_HEADER_IX_MSG	row lock waits	5960
ES_HEADER_IX_DATE	row lock waits	2617
STATS\$SQL_SUMMARY_PK	row lock waits	2467

And we can answer the question 'Are all of the buffer busy waits on index segments?' with the query:

```
SQL> select owner, object_name, object_type, statistic_name, value
2  from v$segment_statistics
3  where statistic_name='buffer busy waits'
4  order by value desc;
```

OBJECT_NAME	OBJECT_TYPE	STATISTIC_NAME	VALUE
ES_INSTANCE_IX_TYPE	INDEX	buffer busy waits	1615317
OM_QUEUE_KEY	INDEX	buffer busy waits	114935
ES_RECIPIENT_IX_MSG	INDEX	buffer busy waits	112891
ES_SHELL	TABLE	buffer busy waits	109969
ES_SHELL_IX_MSG	INDEX	buffer busy waits	94447
ES_QUEUE_IX	INDEX	buffer busy waits	86575
ES_ENVELOPE_IX_MSG	INDEX	buffer busy waits	70724
ES_QUEUE	TABLE	buffer busy waits	52527

### REAL TIME EXECUTION PLANS (ORACLE9IR1)

The Explain Plan facility has been available in Oracle for as long as there has been an optimizer. Although it provides a very useful capability it does have some limitations. With so many factors affecting the outcome of the optimization phase it is not always easy to run explain plan in exactly the same environment as the actual operation of the statement. One example of this is the use of bind values by the optimizer in Oracle9i; bind values are always given default selectivity when using explain plan. The fixed table *v\$sql\_plan*, introduced in Oracle9i allows the DBA to determine the plan for statements that are in the library cache. To join to *v\$sql\_plan* use the hash\_value and address from *v\$sql*. *V\$sql* has been updated to include a plan\_hash\_value. Note that not all statements will have a plan\_hash\_value, or indeed a plan; PL/SQL blocks and insert into <table> values ( ) will both have a plan hash of 0. Now finding the plan for the SQL statement with a hash value of 3291849396 that has been executing for the last 35 minutes is as simple as:

```
SQL> select * from v$sql_plan where hash_value =3291849396;
```

Example output from *v\$sql\_plan* can be seen below in the new Statspack SQL report sprepsql output.

Finding statements that use embedded literal values is now easily achieved by detecting many statements with the same plan hash. This query finds plan\_hash\_values that have the most associated statements:

```
SQL> select min(hash_value), max(hash_value), plan_hash_value, count(*) from v$sql
2  where plan_hash_value > 0
3  group by plan_hash_value
4  order by count(*) desc;
```

MIN(HASH_VALUE)	MAX(HASH_VALUE)	PLAN_HASH_VALUE	COUNT(*)
10701726	4287969013	3704668514	212
27763764	4283196539	3543395131	149
8276870	4246355443	15073971	127
17156281	4162063114	2954689482	58
90885655	4126719415	3119524502	53
1132379143	1132379143	2849082594	49
176245406	3848387593	3549230955	30

3393467370

4151723123

833584900

22

This query also identifies a statement that has only a single hash value but has multiple copies; hash value 1132379143 has 49 copies. Further investigation of this statement was able to determine that the reason for multiple copies was bind mismatch, using the following SQL:

```
SQL> select * from v$sql_shared_cursor
      2  where kglhdpar=(select max(address) from v$sql where hash_value = 1132379143)
```

### ADVISORIES

The intention of the advisories is to provide a straightforward set of data that quantifies the effects that will result from a change in database instance configuration. Advisories work by simulating a range of different configuration sizes. The three advisories available in Oracle9iR2 are:

- Buffer cache
- Shared pool
- PGA memory

The first advisory, for the buffer cache, `db_cache_advice` was introduced in Oracle8i Database Release 1. The advisory data is presented in the fixed table `v$db_cache_advice`, which indicates the number of IOs that the simulation predicts would result from running the same system and workload with 10 different cache sizes ranging from 10% - 200% of the current size. This allows a DBA to determine the optimal buffer cache size for the workload.

A DBA can now determine how many IOs would be saved by a 10% increase in the buffer cache by issuing the statement:

```
SQL> select size_for_estimate cache_size, buffers_for_estimate buffers, size_factor,
      3  estd_physical_read_factor read_fact, estd_physical_reads estd_reads
      2  from v$db_cache_advice;
```

CACHE_SIZE	BUFFERS	SIZE_FACTOR	READ_FACT	ESTD_READS
256	30608	.6957	1.3948	346011
288	34434	.7826	1.2273	304458
320	38260	.8696	1.1096	275262
352	42086	.9565	1.0334	256355
368	43999	1	1	248070
384	45912	1.0435	.975	241879
416	49738	1.1304	.9304	230794
448	53564	1.2174	.8992	223074
480	57390	1.3043	.8695	215688
512	61216	1.3913	.8487	210525

The two new advisories in Oracle9i Database Release 2 provide assistance in setting the shared pool (more accurately the library cache) and PGA memory, for workarea sizing by use of `v$shared_pool_advice` and `v$pga_target_advice` respectively.

The Shared Pool Advisor tracks the library cache's use of shared pool memory and predicts the change in total instance-wide parse time for different sizes of the shared pool. Two new views, `v$shared_pool_advice` and `v$library_cache_memory` provide the information to determine how much memory the library cache is using, how much is currently pinned, how much is on the shared pool's LRU list, as well as how much time might be lost or gained by changing the size of the shared pool. The `v$shared_pool_advice` view displays information about estimated parse time savings in different sizes of shared pool. The sizes range from 50% to 200% of current shared pool size, in equal intervals.

Similar to the Shared Pool Advisor, the PGA Advisor also uses internal simulations to predict the optimal PGA size for the current workload. The view *v\$pga\_target\_advice* contains the results of these simulations showing the appropriate setting for the initialization parameter *pga\_aggregate\_target*.

### V\$SQL ENHANCEMENTS

The contents of the library cache are externalized primarily through the table *v\$sql*. This table has had many columns added since its introduction, and the Oracle9i Database adds more. The *plan\_hash\_value* columns has already been described above. Other columns are added to help identify long running and resource intensive SQL statements; we now have CPU and elapsed time data on per statement basis. Previously it has been necessary to use columns such as number of buffer gets and the number of disk\_reads as proxies for these resource usage characteristics.

```
SQL> select hash_value, executions,
2  round(elapsed_time/1000000,2) ela_secs,
3  round(cpu_time/1000000, 2) cpu_secs from (
4  select * from v$sql order by elapsed_time desc)
5  where rownum < 10;
```

HASH_VALUE	EXECUTIONS	ELA_SECS	CPU_SECS
3102627273	296	37442.69	48.06
751106285	2492	8354.08	29.46
976966401	422	4119.57	7.68
1743040847	430	4119.26	7.3
2869637466	32	1019.32	337.15
4026694086	1219	512.66	57.48
2076521586	253	476.48	193.13
2845979113	644	355.19	77.24
573809690	731	334.07	78.29

Also added to *v\$sql* is a column to indicate the number of fetches performed on the cursor. Used in conjunction with the execution count already in *v\$sql* this gives a simple way for the DBA to identify bulk fetch operations that are not making good use of the array interface.

```
SQL> select sql_text, parse_calls, executions, fetches, rows_processed from v$sql
2  where rows_processed > 1000
3  and fetches >= rows_processed
4  and rows_processed/executions > 5
5  and executions > 0;
```

SQL\_TEXT

PARSE_CALLS	EXECUTIONS	FETCHES	ROWS_PROCESSED
815	815	5911	5911

### UNDO STATISTICS

The table *v\$undostat* appeared with the introduction of Automatic Undo Management in Oracle9i Database Release 1. This table provides a 'recent history' of system activity with one row for every 10 minutes. This allows the DBA to get a picture of the activity of the system over time. Amongst other data it provides information on the rate of undo generation, which is a useful metric for the real work done by a transactional system, and on the numbers of concurrent transaction and longest query length.

```
SQL> select to_char(end_time,'hh24:mi') end_time, undoblks, txncount, maxquerylen,
maxconcurrency from v$undostat;
```

END_T	UNDOBLKS	TXNCOUNT	MAXQUERYLEN	MAXCONCURRENCY
10:44	55	97103	2141	2
10:41	194	96765	15	2
10:31	193	95237	7	2
10:21	141	93759	24	2
10:11	159	92530	10	2
10:01	206	91016	1920	1
09:51	304	89946	16	2
09:41	260	88542	23	2
09:31	254	87149	23	2

Additionally 'snapshot too old' errors now write information about the statement that failed into the alert log, greatly aiding diagnosis. Here is an example of what you will see in the log:

```
ORA-01555 caused by SQL statement below (Query Duration=23261 sec,
      SCN: 0x0578.83ed6d7b):
Wed Oct 2 19:16:24 2002
SELECT count(1) FROM es_header h, es_body b
      WHERE b.msg_id = h.msg_id AND h.internal_date < sysdate - :b1
```

### ENQUEUE STATISTICS

The table *v\$enqueue\_stat* was introduced in Oracle9i Database Release 1. Previously enqueue data was only available through an X\$table. This data was severely limited in that it only captured time for enqueues when the timeout was reached. This made it difficult to drill down when enqueues were a major wait event in the system as often only a small amount of the time spent waiting would be reported against any of the specific enqueue types. The table now contains a column called *cum\_wait\_time*, which is the total wait time (in milliseconds) for each enqueue type.

```
SQL> select * from v$enqueue_stat where cum_wait_time > 0 order by cum_wait_time desc;
```

INST_ID	EQ	TOTAL_REQ#	TOTAL_WAIT#	SUCC_REQ#	FAILED_REQ#	CUM_WAIT_TIME
2	TX	6770534	104518	6240316	530216	249508133
2	US	73043	33809	73043	0	2279519
2	SQ	109252	3405	109252	0	1735823
2	UL	198511	1710	198502	9	1598105
2	TT	264265	139217	264265	0	1141886
2	TA	6495	4319	6490	5	370131
2	CF	1439211	45231	1439207	4	354739
2	HW	102883	24357	102849	34	131230

## **STATSPACK IMPROVEMENTS**

### *ADDITIONAL DATA CAPTURE LEVELS*

Statspack captures almost all of the new database statistics detailed above. Two new levels of data capture are added:

- Snap\_level=6 - captures execution plans for all SQL statements captured in the snapshot
- Snap\_level=7 - captures segment statistics based on thresholds.

To minimize the storage needed by the Statspack schema, both the execution plans and the segment names are normalized into tables separate from the usage data, so that a complex plan that is in use at each snapshot will be stored only once. For the segment statistics, the data is pivoted so that only a single row is captured per segment.

### *MODIFIED TOP EVENTS SECTION*

The Top 5 Wait events section on the first page of the Statspack report (spreport.sql) has been renamed to Top 5 Timed events as it now includes CPU usage as well as wait events. This avoids the need to look at the *v\$sysstat* section of the report to determine if CPU usage or wait events are the main time consumers on a system. Note: this does not mean that there is a CPU time wait event!

<b><u>Top 5 Timed Events</u></b>			
<b>Event</b>	<b>% Total Waits</b>	<b>Time (s)</b>	<b>Ela Time</b>
log file sync	182,574	4,176	45.67
db file sequential read	85,797	1,402	15.34
CPU time		1,116	12.21
db file parallel write	20,718	571	6.24
log file parallel write	171,593	318	3.48

*NEW SEGMENT STATISTICS SECTION*

The report has also been enhanced to output the new segment data. Segment statistics are reported as a new section, similar to the Top SQL section, which gives Top 5 Segments by

- Logical reads
- Physical reads
- Buffer busy waits
- Row lock waits
- ITL waits
- Blocks served (Real Application Clusters)

This section makes it simple for the DBA to identify segments that are receiving heavy load or contributing to the top wait events in the system.

```

Top 5 Segments by Logical Reads for DB: B01 Instance: B01_1 Snaps: 1336 -1337
-> End Segment Logical Reads Threshold: 10000

```

Owner	Tablespace	Object Name	Subobject Name	Obj. Type	Logical Reads	%Total
SAPR3	PSAPSTABD	UST12		TABLE	705,056	7.22
SAPR3	PSAPSTABI	MARA~0		INDEX	295,920	3.03
SAPR3	PSAPSTABI	MAKT~0		INDEX	295,392	3.02
SAPR3	PSAPSTABI	MARC~0		INDEX	292,928	3.00
SAPR3	PSAPSTABI	KONP~0		INDEX	260,656	2.67

```

Top 5 Segment by Physical Reads for DB: B01 Instance: B01_1 Snaps: 1336 -1337
-> End Segment Physical Reads Threshold: 1000

```

Owner	Tablespace	Object Name	Subobject Name	Obj. Type	Physical Reads	%Total
SAPR3	PSAPSTABD	UST12		TABLE	2,709	45.84
SAPR3	PSAPSTABI	UST12~VON		INDEX	523	8.85
SAPR3	PSAPPOOLD	ATAB		TABLE	430	7.28
SAPR3	PSAPPOOLI	ATAB~0		INDEX	291	4.92
SAPR3	MBEW_D	MBEW		TABLE	149	2.52

```

Top 5 Segments by Buf. Busy Waits for DB: B01 Instance: B01_1 Snaps: 1336 -1337
-> End Segment Buffer Busy Waits Threshold: 100

```

Owner	Tablespace	Object Name	Subobject Name	Obj. Type	Buffer Busy Waits	%Total
SAPR3	M_VMVL_I	M_VMVL~0		INDEX	537	15.05
SAPR3	KOCLU_I	KOCLU~0		INDEX	374	10.49
SAPR3	PSAPPOOLD	ATAB		TABLE	231	6.48
SAPR3	RFBLG_I	RFBLG~0		INDEX	199	5.58
SAPR3	VB1X	VBDATA400		TABLE	131	3.67

**NEW STATSPACK SQL REPORT**

The execution plan data captured from `v$sql_plan` is only visible via the new Statspack SQL report (sprepsql.sql).

This will report on the use of a single SQL statement between two snapshots and will give a history of the plans that have been used by the statement. The report also includes the full text of the statement and the resources used between the specified snapshots. Below is an example.

STATSPACK SQL report for Hash Value: 2469923189					Module: SQL*Plus	
<u>DB Name</u>	<u>DB Id</u>	<u>Instance</u>	<u>Inst Num</u>	<u>Release</u>	<u>Cluster</u>	<u>Host</u>
9IR2	1460794412	9ir2	1	9.2.0.1.0	NO	dlsun4216
<u>Start Id</u>	<u>Start Time</u>	<u>End Id</u>	<u>End Time</u>	<u>Duration(mins)</u>		
6	01-May-02 20:09:24	7	01-May-02 20:21:14	11.83		
<b><u>SQL Statistics</u></b>						
-> CPU and Elapsed Time are in seconds (s) for Statement Total and in milliseconds (ms) for Per Execute						
		Statement Total	Per Execute	% Snap Total		
		-----	-----	-----		
Buffer Gets:		2,758,825	2,758,825.0	99.84		
Disk Reads:		39,336	39,336.0	97.71		
Rows processed:		0	0.0			
CPU Time(s/ms):		697	697,070.0			
Elapsed Time(s/ms):		684	684,111.2			
Sorts:		1	1.0			
Parse Calls:		1	1.0			
Invalidations:		0				
Version count:		2				
Sharable Mem(K):		56				
Executions:		1				
<b><u>SQL Text</u></b>						
Select calc.MDA_DIM2_ID, calc.MDA_DIM1_ID, calc.MDA_DIM3_ID, ca ...						
<b><u>All Optimizer Plan(s) for this Hash Value</u></b>						
Shows all known Optimizer Plans for this Hash value, and the Snap Id's they were first found in the shared pool -> ordered by Snap Id						
	Plan					
Hash Value	Snap Id	Cost	Optimizer			
-----	-----	-----	-----			
3767997828	4	0	RULE			
511062383	7	1457	CHOOSE			
3767997828	7	0	RULE			

Plans in shared pool between Begin and End Snap Ids

-> Rows indicates Cardinality, PHV is Plan Hash Value  
-> ordered by Plan Hash Value

<u>Operation</u>	<u>PHV/Object Name</u>	<u>Rows</u>	<u>Bytes</u>	<u>Cost</u>
SELECT STATEMENT	----- 511062383 ----			1457
SORT GROUP BY		1	32	1457
FILTER				
TABLE ACCESS BY INDEX ROWID	PS_MD_DIFF_T4	1	15	4
NESTED LOOPS		1	32	1455
TABLE ACCESS FULL	PS_MD_CALC_T4	1	17	1451
INDEX RANGE SCAN	PS_MD_DIFF_T4	1		3
SORT AGGREGATE		1	15	
FIRST ROW		1	15	3
INDEX RANGE SCAN (MIN/MAX)	PSAMD_CALC_T4	390K		3
SORT AGGREGATE		1	16	
FIRST ROW		1	16	3
INDEX RANGE SCAN (MIN/MAX)	PSAMD_CALC_T4	390K		3
SELECT STATEMENT	----- 3767997828 ---			
SORT GROUP BY				
FILTER				
TABLE ACCESS BY INDEX ROWID	PS_MD_CALC_T4			
NESTED LOOPS				
TABLE ACCESS FULL	PS_MD_DIFF_T4			
INDEX RANGE SCAN	PSAMD_CALC_T4			
SORT AGGREGATE				
INDEX RANGE SCAN	PSAMD_CALC_T4			
SORT AGGREGATE				
INDEX RANGE SCAN	PSAMD_CALC_T4			

End of Report

*ADDITIONAL REPORTING CHANGES*

- CPU and elapsed time data from *v\$sql* in Top SQL section
- Buffer cache, SGA and PGA memory advisory tables are reported

## TWEAKS AND FEATURETTES

This is a list of small changes that have been made in the database that are very likely to be under the radar of most DBAs. Enjoy!

- Wait events and System Statistics
  - Microseconds timer used for all wait events rather than 10 millisecond
  - IOs and buffer busy waits now populate the *n\$session.row\_wait\_obj#* column allowing easy determination of the object that is the subject of IOs or buffer busy waits.
  - Parse count (failures) statistic
  - Workarea execution statistics indicating workareas spilling to disk
  
- SQL trace
  - Microsecond timer for CPU and elapsed times rather than 10 millisecond
  - Tkprof now outputs data about wait events for each SQL statement if they are present in the SQL trace file
  - 'Unsupported' dbms\_support (dbmssupp.sql) package for enabling sql\_trace with wait events and bind value options
    - e.g. `begin dbms_support.start_trace_in_session(102,13,waits=>TRUE, binds=>FALSE); end;`
  
- Explain plan
  - Plan entries now contain access and filter predicates on each row in the plan
  - Search\_columns indicates how many columns of an index are used
  - Dbms\_xplan package (in dbmsutil.sql) to simplify SQL execution plan reporting
  
- Lock monitoring
  - Improved performance of dba\_waiters and dba\_blockers

## CONCLUSIONS

Performance diagnostics was one of the key development focus areas for Oracle9i. Numerous enhancements were made in the database server to aid DBAs in the performance diagnostic process. The new memory tuning advisors, several new diagnostic statistics, improvements in the Statspack package, are just few of the areas where Oracle9i Database provides a tremendous advantage. The new statistics that have been added to the Oracle9i Database have been designed to aid the DBA and make diagnosing issues simpler.