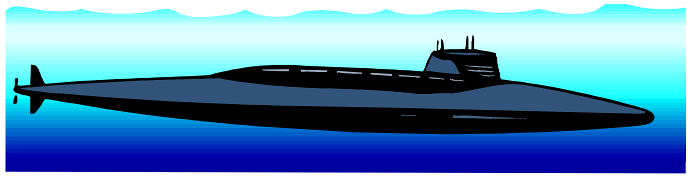
Ten Things you May Miss in an AWR that are Robbing Performance

Mike Ault

Oracle Flash Consulting Manager

IBM, Corp.

Michael R. Ault, Oracle Guru



Nuclear Navy 6 years

Nuclear Chemist/Programmer 10 years

Bachelors Degree Computer Science

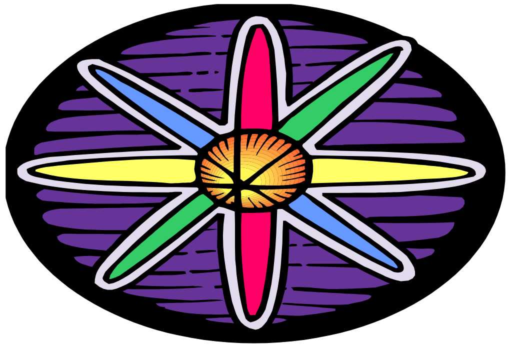
Certified in all Oracle Versions Since 6

Oracle DBA, author, since 1990

Oracle ACE

ORACLE ACE

ORACLE CERTIFIED PROFESSIONAL



Introduction

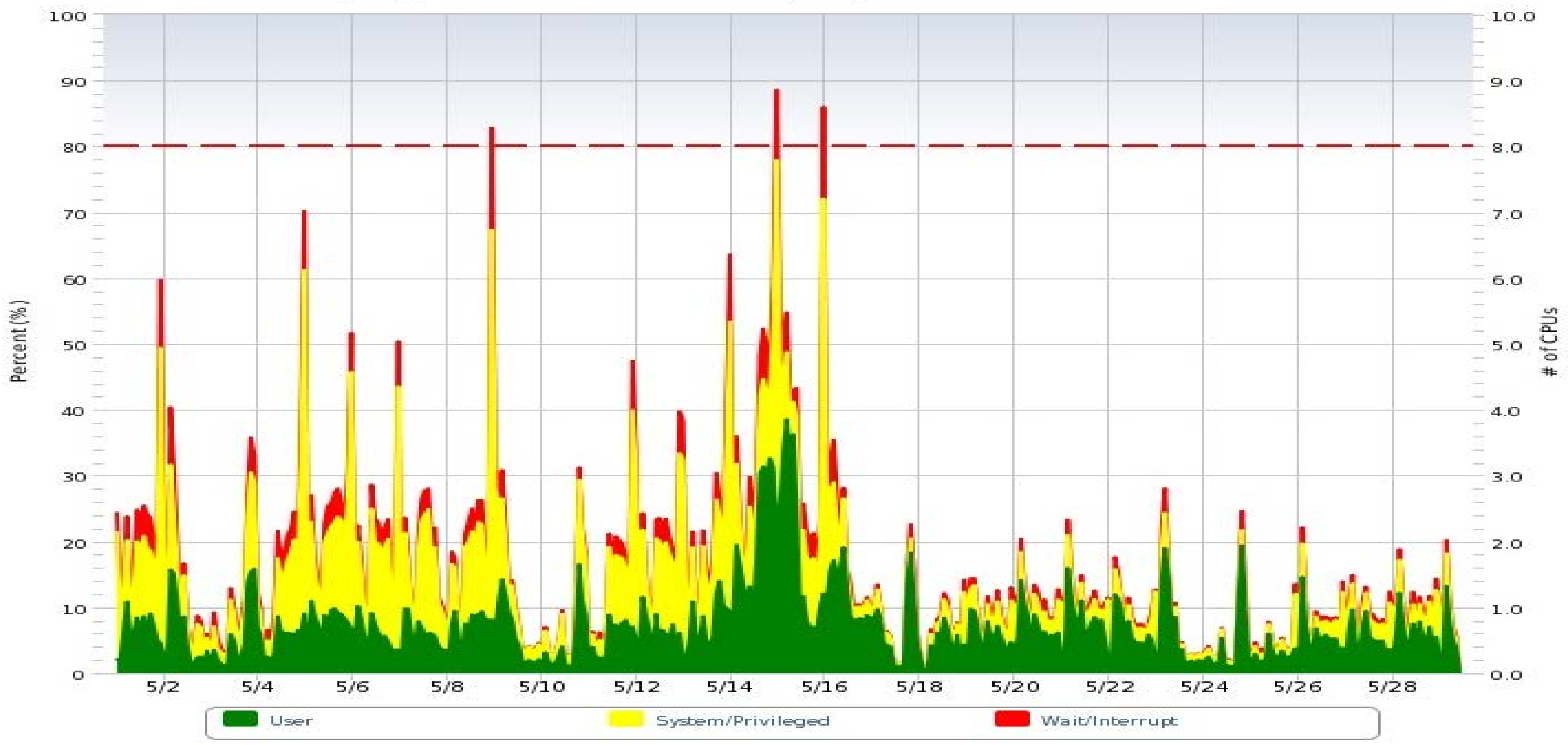
- Analyzed bstat/estat, Statspack and AWR since they were invented
- Many constants
  - Lack of use of Bind Variables
  - Improper memory use
  - Use of default initialization parameters
  - Leapfrog development cycles
    - Reoccurring bugs
- Let’s look at some of the top 10 issues I see

Not Setting the FILESYSTEMIO_OPTIONS

- I/O operations typically go through the file system cache.
  - Extra processing requires CPU resources.
- Bypassing the file system cache:
  - Reduces CPU requirements
  - Frees up the file system cache for other non-database file operations.
  - Raw devices automatically bypass the file system cache.
- Synchronous I/O requests
  - blocks until the write is complete before continuing processing.
- Asynchronous I/O requests
  - Processing continues while the I/O request is submitted and processed.
  - Asynchronous I/O bypasses the performance bottlenecks with I/O operations.
- Use direct I/O and asynchronous I/O using the **FILESYSTEMIO_OPTIONS** parameter
  - ASYNCH - Enabled asynchronous I/O where possible.
  - DIRECTIO- Enabled direct I/O where possible.
  - SETALL- Enabled both direct I/O and asynchronous I/O where possible.
  - NONE - Disabled both direct I/O and asynchronous I/O.

Not Setting FILESYSTEMIO_OPTIONS

- Setting FILESYSTEMIO_OPTIONS properly can boost performance by up to 30%
- Setting it properly also reduces CPU spikes.
- Oracle defaults it to NULL or NONE
- If it isn't listed the initialization parameter section, it isn't set properly.



Caveats

- FILESYSTEMIO_OPTIONS is ignored with ASM
- In ASM make sure DISK_ASYNC_IO is set to TRUE (default)

The Segment Reports

- Segment Sections show segments by the type of wait action most seen.
  - Physical IO
  - Direct IO
  - Full Table Scans
  - Un-optimized IO
  - Several RAC related issues
- Analyze the segments in these sections toquickly see the majority of IO
- One issue these reports don’t do a rollup for partitioned segments
  - Large segments with partitions can dominate report sections
- Review the physical IO related reports for segments that can benefit from indexes.
- If an index shows that can be a good thing or a bad thing, but it shows you where your IO dollar is being spent in the database.

Segment Statistics in Oracle12c

- Segments by Logical Reads
- Segments by Physical Reads
- Segments by Physical Read Requests
- Segments by UnOptimized Reads
- Segments by Optimized Reads
- Segments by Direct Physical Reads
- Segments by Physical Writes
- Segments by Physical Write Requests
- Segments by Direct Physical Writes
- Segments by Table Scans
- Segments by DB Blocks Changes
- Segments by Row Lock Waits
- Segments by ITL Waits
- Segments by Buffer Busy Waits

Example Segment Report

Segments by Physical Reads

•Total Physical Reads: 1,930,066

•Captured Segments account for 87.2% of Total

Owner	Tablespace Name	Object Name	Subobject Name	Obj. Type	Physical Reads	%Total
SAPSR3	PSAPSR3	LDQ_STATE		TABLE	931,268	48.25
SYS	SYSTEM	COL$		TABLE	139,723	7.24
SAPSR3	PSAPSR3	LDQ_DATA		TABLE	90,156	4.67
SAPSR3	PSAPSR3	LDQ_STATE~1		INDEX	73,438	3.80
SAPSR3	PSAPSR3	SMMW_MSG_HDR		TABLE	61,625	3.19

Temporary Activity to Disk

- Temporary activity includes:
  - Sorts
  - Hashes
  - Global Temporary Table Overflow
  - Bitmap Operation overflow
    - Create
    - Merge
- In 10g PGA_AGGREGATE_TARGET automated temporary segment processing
- Use of shared server negates automated temporary segment processing
  - Oracle uses old parameters
    - SORT_AREA_SIZE – default 64 mb
    - HASH_AREA_SIZE – default 2X SORT_AREA_SIZE
    - CREATE_BITMAP_AREA_SIZE – default 8 mb
    - BITMAP_MERGE_AREA_SIZE– default 1 mb

Temporary Activity to Disk

- How is shared servers turned on?
  - By default
    - Oracle sets DISPATCHERS – defaults to a derived name ending in XDB
    - Oracle sets SHARED_SERVERS – defaults to 1 (not shown in parameter listing)
  - Oracle development tools use the DISPATCHER that is created to connect
  - After development is over turn them off
  - If you don’t use Oracle tools turn them off

Signs Shared Server is turned on

Shared Servers Rates

Common Queue Per Sec	Disp Queue Per Sec	Server Msgs/Sec	Server KB/Sec	Common Queue Total	Disp Queue Total	Server Total Msgs	Server Total(KB)
0	0	0	0.00	0	0	0	0

Shared Servers Utilization

•Statistics are combined for all servers
•Incoming and Outgoing Net % are included in %Busy

Total Server Time (s)	%Busy	%Idle	Incoming Net %	Outgoing Net %
3,689	0.00	100.00	0.00	0.00

Shared Servers Dispatchers

•Ordered by %Busy, descending
•Total Queued, Total Queue Wait and Avg Queue Wait are for dispatcher queue
•Name suffixes: "(N)" - dispatcher started between begin and end snapshots "(R)" - dispatcher re-started between begin and end snapshots

Name	Avg Conns	Total Disp Time (s)	%Busy	%Idle	Total Queued	Total Queue Wait (s)	Avg Queue Wait (ms)
D000	0.00	3,689	0.00	100.00	0	0	

Symptoms in AWR of Temporary Segments to Disk

- Temporary tablespace one of top sources of IO

Tablespace IO Stats

•ordered by IOs (Reads + Writes) desc

Tablespace	Reads	Av Reads/s	Av Rd(ms)	Av Blks/Rd	Writes	Av Writes/s	Buffer Waits	Av Buf Wt(ms)
TST_AUTO_TBL03	2,610,787	723	0.96	12.13	1,272,803	353	27,228	1.05
TST_TEMP	262,510	73	0.00	3.02	1,616,935	448	0	0.00
TST_AUTO_TBL04	814,444	226	0.05	31.12	3,678	1	55	5.64
LRX_LRG_T01	447,172	124	0.01	31.93	88	0	0	0.00
TST_RULE_3_PARTIX07	77,400	21	7.17	1.00	70,278	19	0	0.00
TST_RULE_3_PARTIX08	77,578	21	7.10	1.00	70,076	19	0	0.00

Symptoms in AWR of Temporary Segments to Disk

- Statistics in Instance Statistics section

Instance Activity Stats

•Ordered by statistic name

Statistic			
sorts (disk)	3	0.00	0.01
workarea executions - onepass	6,542	1.81	20.83
workarea executions - multipass	100	0	0

Symptoms in AWR of Temporary Segments to Disk

- PGA Aggr Target istogram shows segments <512 mb

PGA Aggr Target Histogram

•Optimal Executions are purely in-memory operations

Low Optimal	High Optimal	Total Execs	Optimal Execs	1-Pass Execs	M-Pass Execs
2K	4K	9,315	9,315	0	0
64K	128K	89	89	0	0
128K	256K	1,081	1,081	0	0
256K	512K	1,723	1,723	0	0
512K	1024K	926	926	0	0
1M	2M	3,520	3,520	0	0
2M	4M	5,987	5,733	254	0
4M	8M	2,983	2,983	0	0
8M	16M	5,490	368	5,122	0
16M	32M	1,221	90	1,131	0
32M	64M	41	39	2	0
64M	128M	22	20	2	0
128M	256M	2	0	2	0
256M	512M	2	2	0	0

Why is 512 MB Important?

- PGA_AGGREGATE_TARGET is used to control overall memory assigned for temporary actions
- Each process gets 5% up to maximum set by _PGA_MAX_SIZE
- _PGA_MAX_SIZE defaults to 512 mb
- The automated process should handle temporary activity below 512 MB
- If you see temporary activity less than 512 mb going to storage this is out-of-band (OOB)

How do you know PGAT Is Right?

- PGA Memory Advisor

PGA Memory Advisory

•When using Auto Memory Mgmt, minimally choose a pga_aggregate_target value where Estd PGA Overalloc Count is 0

PGA Target Est (MB)	Size Factr	W/A MB Processed	Estd Extra W/A MB Read/ Written to Disk	Estd PGA Cache Hit %	Estd PGA Overalloc Count	Estd Time
5,000	0.13	133,611,373.09	66,289,825.32	67.00	426	125,932,781,781
10,000	0.25	133,611,373.09	51,376,357.58	72.00	20	116,537,668,132
20,000	0.50	133,611,373.09	43,447,055.91	75.00	0	111,542,405,347
30,000	0.75	133,611,373.09	43,186,367.60	76.00	0	111,378,178,196
40,000	1.00	133,611,373.09	37,192,946.09	78.00	0	107,602,471,753
48,000	1.20	133,611,373.09	10,565,649.96	93.00	0	90,827,937,632

Versioning in SQL

- Looking at the header sections of AWR
- Default DB cache should be biggest area
- If Shared Pool>= Default DB cache – then SQL issues
- One recurrent issue is SQL versioning

Versioning in SQL

- When a SQL is issued Oracle assigns it an ID based on a signature of the entire length of the SQL
- The optimizer also determines if it contains
  - AND
  - OR
  - BETWEEN
  - IN
- If it matches an already existing ID then it looks to see what the bind variables contain and assigns a cost based on what it thinks it will require to fulfill the request
- If the cost it too much different it will assign a new version/child cursor and a new parse tree and execution path
- This takes up memory in shared pool
- A bug that started in Oracle10 causes excessive versioning

Example of Versioning in 11.2

SQL ordered by Version Count

•Only Statements with Version Count greater than 20 are displayed

Version Count	Executions	SQL Id	SQL Module	SQL Text
2,349	905	bvcy008mdghs9	TSTR6-RAD	SELECT * FROM (SELECT TRIM (G...
2,349	905	bvcy008mdghs9	TSTR6-RAD	SELECT * FROM (SELECT TRIM (G...
2,349	905	bvcy008mdghs9	TSTR6-RAD	SELECT * FROM (SELECT TRIM (G...
2,278	66	26vkzugxydvvy	TSTRv4-RAI	SELECT NUMERO, LIBELLE, NUMC...
2,278	66	26vkzugxydvvy	TSTRv4-RAI	SELECT NUMERO, LIBELLE, NUMC...
2,278	66	26vkzugxydvvy	TSTRv4-RAI	SELECT NUMERO, LIBELLE, NUMC...
1,345	3,137	ansxkng63hgf5	TSTR6-RAI	SELECT NUMERO, LIBELLE, NUMC...
1,345	3,137	ansxkng63hgf5	TSTR6-RAI	SELECT NUMERO, LIBELLE, NUMC...
1,345	3,137	ansxkng63hgf5	TSTR6-RAI	SELECT NUMERO, LIBELLE, NUMC...
1,345	3,137	ansxkng63hgf5	TSTR6-RAI	SELECT NUMERO, LIBELLE, NUMC...

Report Summary

Cache Sizes

	Begin	End		
Buffer Cache:	10,112M	10,240M	Std Block Size:	32K
Shared Pool Size:	12,928M	12,928M	Log Buffer:	91,424K

The fix for Versioning

- Set the undocumented parameter “_sqlexec_progression_cost” to its maximum value
- Turns off versioning

Thrashing

- Thrashing: one memory area passes memory chunks back and forth with one or more other memory areas
- Can only happen when automatic memory management is being used

Memory Resize Operations Summary

•Resizes, Grows, Shrinks - Operations captured by AWR if there are operations on the same component for the same operation_type, target_size, and with the same start_time only one operation is captured
•ordered by Component

Component	Min Size (Mb)	Max Size (Mb)	Avg Size (Mb)	Re- Sizes	Grows	Shrinks
DEFAULT buffer cache	14,920.00	15,920.00	15,549.55	349	170	179
shared pool	5,532.00	6,532.00	5,902.45	349	179	170

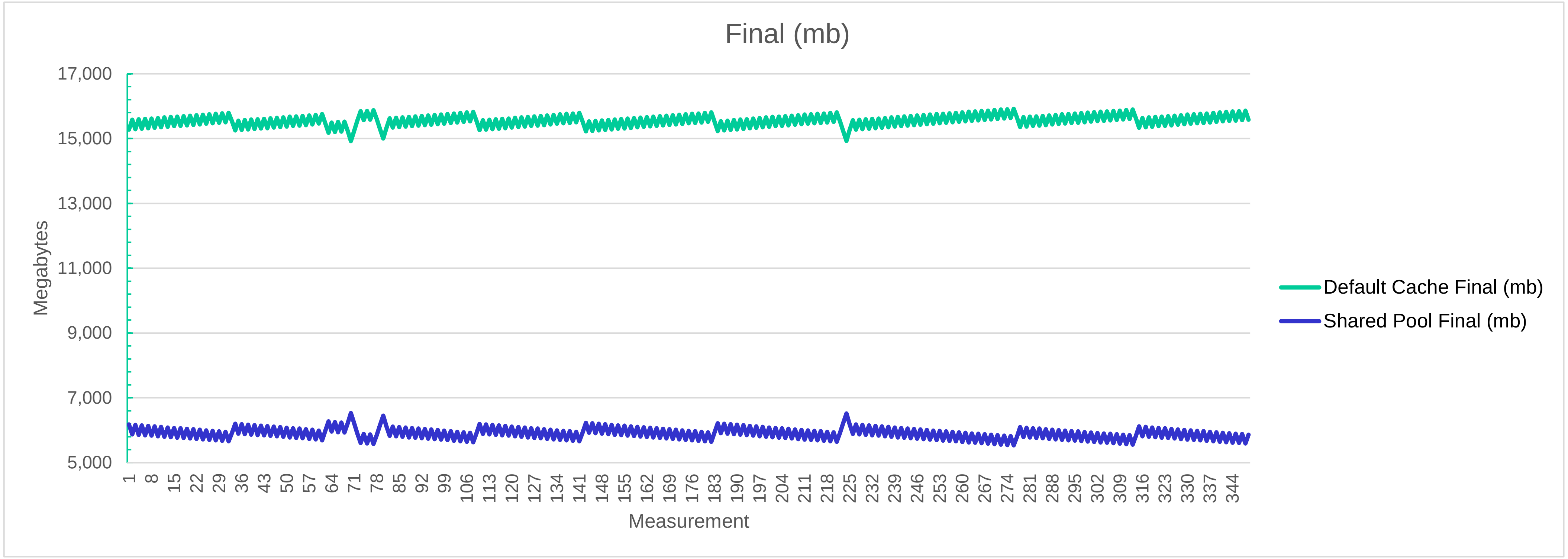
Memory Resize Ops

•Oper Types/Modes: INItializing,GROw,SHRink,STAtic/IMMediate,DEFerred Delta : change in size of the component Target Delta: displayed only if final size <> target_size
•Status: COMplete/CANcelled/INActive/PENding/ERRor
•ordered by start_time desc,component

Start	Ela (s)	Component	Oper Typ/Mod	Init Size (M)	Delta	Target Delta	Final (M)	Sta
03/07 11:47:40	1	bufcache	SHR/DEF	15,860	-276		15,584	COM
03/07 11:47:40	1	shared	GRO/DEF	5,592	276		5,868	COM
03/07 11:46:40	0	bufcache	GRO/DEF	15,568	292		15,860	COM
03/07 11:46:40	0	shared	SHR/DEF	5,884	-292		5,592	COM
03/07 10:09:05	1	bufcache	SHR/DEF	15,848	-280		15,568	COM
03/07 10:09:05	1	shared	GRO/DEF	5,604	280		5,884	COM

Thrashing: The issue, cache misses

- A DB cache miss just generates a couple of IOs
- A SQL Shared Pool cache miss causes a hard-parse
- Hard parses are expensive
- Excessive CPU can be traced back to excessive SQL shared pool misses



Thrashing: The Fix

- Increase the SGA and MEMORY sizing parameters
  - SGA_MAX_SIZE – Max SGA size for instance lifetime
  - SGA_TARGET – Starting SGA size at initial startup (initial total of all components)
    - DBA cache
    - Shared Pool
    - Large pool
    - Java Pool
    - Streams pool
  - MEMORY_MAX_TARGET – Maximum size of SGA plus PGA_AGGREGATE_TARGET
  - MEMORY_TARGET – Starting SGA size plus starting PGA_AGGREGATE_TARGET
    - DBA cache
    - Shared Pool
    - Large pool
    - Java Pool
    - Streams pool
    - PGA_AGGREGATE_TARGET
  - Also set base size of internal components
  - You set SGA_MAX=SGA_TARGET or MEMORY_MAX=MEMORY_TARGET you will get thrashing

Round Trips

- Your spouse asks you to get 6 bananas
  - You go to the store, buy one banana
  - You return home, drop off banana
  - You go back to store, get second banana
  - Return home drop off banana
  - Repeat until number of desired bananas are purchased
  - DOES THIS MAKE SENSE?
- SQLNet, Java and other interfaces are limited in array fetch size for cursors
  - SQLNet – 10 results
  - Java – 15 results
- So, if you have 1500 results SQLNet will do 150 round trips and Java 100
- The database will report sub-second response, the user will see multi-second response, both will be right
- Limit roundtrips to less than 100 by increasing array passing sizes and tuning the network

Instance Activity Stats

Statistic	Total	per Second	per Trans
SQL*Net roundtrips to/from client	1,501,575	833.28	683.16

Physical IO the root of all evil…

- Well, at least in a database!
- Rich Niemiec, past president of TUSC and current President of ROLTA said a majority of Oracle performance issues are IO related
- In analyzing hundreds of reports I have to agree

Load Profile

	Per Second	Per Transaction
Physical reads:	5,101.28	4,182.21
Physical writes:	5,284.01	4,332.02

- So…we have 10,385 IOPS here right?

IO

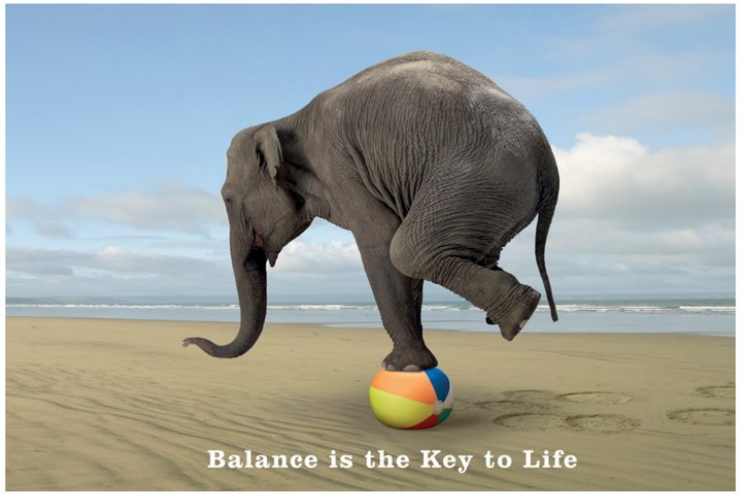
- Nope!

Instance Activity Stats

Statistic	Total	per Second	per Trans
physical read total IO requests	1,448,599	803.88	659.05
physical read total bytes	77,488,745,472	43,001,523.57	35,254,206.31
physical write total IO requests	2,821,755	1,565.90	1,283.78
physical write total bytes	################	55,542,924.08	45,536,100.63

- We have 2,370 IOPS… big difference!
- The Load Section reports Block IOPS, the Instance Activity Stats and IO Stats breakouts report true IOPS
- I’ve had folks tell me they had 20,000 IOPS on a standard disk array because they looked at the wrong statistics

IO Balance



- Ok, so we have IO, but what kind?
- You get a general idea comparing block IOPS with true IOPS
  - If they are close, then mostly single block IO
  - If they are wildly different then multi-block IO
- You can use the IO stats rollups to get general IO size
- The tablespace IO reports also show IO size by tablespace and latency
- You can calculate overall IO size for reads and writes using the byes read and written and the number of reads and writes.
- Pay attention to top 5/top 10 listings!

Top Wait Listings

- Tell you IO distribution
  - DB file sequential read – single block IO
  - Cell Physical Single Block Read – Single block IO
  - DB file scattered read, direct read – Multi-block IO
  - Cell Physical Multi-block read – Multi-block read
  - Log file – 128 K write

Top 10 Foreground Events by Total Wait Time

Event	Waits	Total Wait Time (sec)	Wait Avg(ms)	% DB time	Wait Class
db file sequential read	1,692,121	16.3K	10	89.7	User I/O
log file sync	200,106	689.2	3	3.8	Commit
DB CPU		659.1		3.6	
control file sequential read	57,557	60.4	1	.3	System I/O
read by other session	4,620	53.8	12	.3	User I/O
db file parallel read	8,957	49.5	6	.3	User I/O
recovery area: computing obsolete files	17	20.7	1220	.1	Other
enq: TX - index contention	397	14.1	36	.1	Concurrency
db file scattered read	646	12.7	20	.1	User I/O
direct path read	850	11	13	.1	User I/O

IO

- In the previous listing look at the top event:

Event	Waits	Total Wait Time (sec)	Wait Avg(ms)	% DB time	Wait Class
db file sequential read	1,692,121	16.3K	10	89.7	User I/O

- By mitigating this one event we can recover nearly 90% of db time and give it to the CPU
- Replacing 10 ms storage with 500 microsecond (0.5 ms) we could reduce this by a factor of 20x
- It would also fix the log file issue and the read by other session
- If a system is properly tuned then it uses the minimal amount o all resources to get the desired results
  - It isn’t always possible
  - Then we have to resort to hardware!

When is a Redo Log like a Writing Desk?

- Well, never actually, it just sounded neat
- Redo logs can be a major source of wait
- Usually this will be the *Log File Sync* wait
  - An umbrella wait encapsulating many others
  - Will usually be equal to the Commit rollup section

Top 5 Timed Foreground Events

Event	Waits	Time(s)	Avg wait (ms)	% DB time	Wait Class
DB CPU		26,264		60.19	
db file sequential read	7,534,745	6,240	1	14.30	User I/O
log file sync	3,235,812	3,611	1	8.28	Commit
direct path read	1,513,251	2,273	2	5.21	User I/O
db file parallel read	1,001,005	1,643	2	3.77	User I/O

Foreground Wait Class

Wait Class	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	%DB time
DB CPU			26,264		60.19
User I/O	10,580,023	0	11,676	1	26.76
Commit	3,235,812	0	3,611	1	8.28

Redo Issues

- Excessive commits
  - Hard to fix
  - May be required
- Co-location with tablespaces
  - Move to their own LUN
- Switching too often
  - Increase size of logs
- If none of the above helps – reduce latency

Cursors…foiled again!

- Cursor Processing
  - Cursor is parsed
  - Tables and columns verified
  - Permissions verified
  - Index existence checked
  - Literals/Bind variables checked
  - Statistics checked
  - Execution path determined
- Bind variables still being missed after nearly 20 years
- Can drive versioning if bind variables not used
- If CURSOR_SHARING not used then each statement with different variables is a new statement

CURSOR_SHARING

- Can be used to mitigate bad code that doesn’t use bind variables in cursors
- Not a fix! A Bandaid!
- Ultimate fix is to correct code
  - May not be possible with 3dr party apps
  - Has three settings:
    - EXACT – default behavior
    - SIMILAR – uses statistics and looking at literals
    - FORCE – Always replace literals (least buggy)
  - Can dramatically reduce hard parsing, CPU usage and Shared Pool memory consumption

Does UNDO Matter

- Before automated undo much effort spent sizing undo segments
  - Rollback segments for us curmudgeons
- Now Oracle mostly gets it right
- When looking at the reports if the last two columns of the UNDO sections are zeros then usually things are great.
- If they aren't all zeros then some tuning may be required.

Undo Segment Summary

•Min/Max TR (mins) - Min and Max Tuned Retention (minutes)
•STO - Snapshot Too Old count, OOS - Out of Space count
•Undo segment block stats:
•uS - unexpired Stolen, uR - unexpired Released, uU - unexpired reUsed
•eS - expired Stolen, eR - expired Released, eU - expired reUsed

Undo TS#	Num Undo Blocks (K)	Number of Transactions	Max Qry Len (s)	Max Tx Concurcy	Min/Max TR (mins)	STO/ OOS	uS/uR/uU/ eS/eR/eU
86	584.30	3,605,960	12,313	12	7430.4/7628.8	0/0	0/0/0/0/0/0

Undo Segment Stats

•Most recent 35 Undostat rows, ordered by Time desc

End Time	Num Undo Blocks	Number of Transactions	Max Qry Len (s)	Max Tx Concy	Tun Ret (mins)	STO/ OOS	uS/uR/uU/ eS/eR/eU
18-Mar 04:53	2,428	13,726	11,707	5	7,629	0/0	0/0/0/0/0/0
18-Mar 04:43	645	9,610	11,105	4	7,608	0/0	0/0/0/0/0/0
18-Mar 04:33	1,215	14,448	10,501	5	7,587	0/0	0/0/0/0/0/0
18-Mar 04:23	2,134	25,953	12,313	5	7,564	0/0	0/0/0/0/0/0

Tuning UNDO

- Three things:
  - Number of rollbacks executed
  - Number and size of undo segments
  - Transactions per rollback segment

Tuning UNDO

- Number of rollbacks
  - A client that keeps disconnecting will cause rollbacks
  - CTRL-C out of long running query or transaction
  - Using INSERT-EXCEPTION-UPDATE instead of MERGE
- Number and Size
  - Auto-tuning starts with 10
  - Size is based on size of tablespace and other internal factors
  - Number is incremented when SESSIONS/TRANSACTIONS_PER_ROLLBACK_SEGMENT exceeds current undo segment count
  - Never decrements
  - Increase size by recreating larger tablespace
  - Increase/decrease number by changing TRANSACTIONS_PER_ROLLBACK_SEGMENT

Some Final Observations

- Small change can often be found under seat cushions – Notebooks of Lazarus Long
- **PX Deq Credit: send blkd** enqueues - parallel query is feeding into non-parallel DML.
  - Make sure you have all the prerequisites satisfied for parallel DML
  - Make sure you have explicitly turned on parallel DML
  - Make sure you have partitioning
- **resmgr:cpu quantum** waits - The *resmgr:cpu quantum* waits can be due to a bug in NUMA optimization and is eliminated with the undocumented setting: “_enable_NUMA_optimization”=FALSE.
  - If you are not prey to this bug, be careful your batch jobs to not compete with Oracle’s autojobs which run from 22:00-06:00 causing CPU throttling and *resmgr:cpu quantum* wait because of the default resource plan.

Conclusion

- The AWR report is awash in statistics
- Determining important statistics from fluff is difficult
- In this paper I tried to show some uncommon findings
- They can make a large difference in performance if you fix them

Questions?

Mike Ault

mrault@us.ibm.com