<http://blog.itpub.net/26954807/viewspace-1300697/>

http://www.askmaclean.com/archives/performance-tuning-oracle-awr.html

【性能调优】Oracle AWR报告指标全解析

有同学在看过《Oracle调优鹰眼，深入理解AWR性能报告》的教学视频后急切期待第三讲，但实际是第三讲需要结合大量的原理知识才能充分理解 例如Latch activity 、Undo、Dynamic Resource Master均需要理解其原理才能充分理解。 所以这些AWR的环节将在 Maclean 今后的 系列调优讲座中介绍。对于《Oracle调优鹰眼系列》则会增加本附录，作为对全部Oracle AWR指标的介绍，本附录对于原理理解方面的内容将不多，而更侧重于指标含义的介绍，是对AWR鹰眼讲座的工具文档。

如果你觉得本AWR解析中的哪些指标仍理解不透彻或者讲的不清楚的，可以在本页中留言，谢谢大家的支持。

Hawk Eyes 看AWR的鹰眼= 基础理论夯实+看过500份以上AWR

# 啥是AWR?

## AWR (Automatic Workload Repository)

一堆历史性能数据，放在SYSAUX表空间上，AWR和SYSAUX都是10g出现的，是Oracle调优的关键特性； 大约1999年左右开始开发，已经有15年历史

默认快照间隔1小时，10g保存7天、11g保存8天; 可以通过DBMS\_WORKLOAD\_REPOSITORY.MODIFY\_SNAPSHOT\_SETTINGS修改

DBA\_HIST\_WR\_CONTROL

AWR程序核心是dbms\_workload\_repository包

* @?/rdbms/admin/awrrpt    本实例
* @?/rdbms/admin/awrrpti   RAC中选择实例号

## 谁维护AWR?

主要是MMON(Manageability Monitor Process)和它的小工进程(m00x)

MMON的功能包括:

1. 启动slave进程m00x去做AWR快照
2. 当某个度量阀值被超过时发出alert告警
3. 为最近改变过的SQL对象捕获指标信息

## AWR小技巧

## 手动执行一个快照：

Exec dbms\_workload\_repository.create\_snapshot;

(这个要背出来哦，用的时候去翻手册，丢脸哦 J!)

## 创建一个AWR基线

|  |
| --- |
| Exec DBMS\_WORKLOAD\_REPOSITORY.CREATE\_BASELINE(start\_snap\_id，end\_snap\_id ,baseline\_name);  @?/rdbms/admin/awrddrpt     AWR比对报告  @?/rdbms/admin/awrgrpt       RAC 全局AWR |

## 自动生成AWR HTML报告：

http://www.oracle-base.com/dba/10g/generate\_multiple\_awr\_reports.sql

# 1、报告总结

|  |
| --- |
| WORKLOAD REPOSITORY report for  DB Name DB Id Instance Inst Num Startup Time Release RAC  ------------ ----------- ------------ -------- --------------- ----------- ---  MAC 2629627371 askmaclean.com 1 22-Jan-13 16:49 11.2.0.3.0 YES  Host Name Platform CPUs Cores Sockets Memory(GB)  ---------------- -------------------------------- ---- ----- ------- ----------  MAC10 AIX-Based Systems (64-bit) 128 32 320.00  Snap Id Snap Time Sessions Curs/Sess  --------- ------------------- -------- ---------  Begin Snap: 5853 23-Jan-13 15:00:56 3,520 1.8  End Snap: 5854 23-Jan-13 15:30:41 3,765 1.9  Elapsed: 29.75 (mins)  DB Time: 7,633.76 (mins) |

* Elapsed

Elapsed 为该AWR性能报告的时间跨度(自然时间的跨度，例如前一个快照snapshot是4点生成的，后一个快照snapshot是6点生成的，则若使用@?/rdbms/admin/awrrpt 脚本中指定这2个快照的话，那么其elapsed = (6-4)=2 个小时)，一个AWR性能报告 至少需要2个AWR snapshot性能快照才能生成 ( 注意这2个快照时间 实例不能重启过，否则指定这2个快照生成AWR性能报告 会报错)，AWR性能报告中的 指标往往是 后一个快照和前一个快照的 指标的delta，这是因为累计值并不能反映某段时间内的系统workload。

* DB TIME

DB TIME= 所有前台session花费在database调用上的总和时间：

注意是前台进程foreground sessions

包括CPU时间、IO Time、和其他一系列非空闲等待时间，别忘了cpu on queue time

DB TIME 不等于 响应时间，DB TIME高了未必响应慢，DB TIME低了未必响应快

DB Time描绘了数据库总体负载，但要和elapsed time逝去时间结合其他来。

* Average Active Session

Average Active Session AAS= DB time/Elapsed Time

DB Time =60 min ， Elapsed Time =60 min AAS=60/60=1 负载一般

DB Time= 1min , Elapsed Time= 60 min AAS= 1/60 负载很轻

DB Time= 60000 min，Elapsed Time= 60 min AAS=1000  系统hang了吧？

DB TIME= DB CPU + Non-Idle Wait + Wait on CPU queue

如果仅有2个逻辑CPU，而2个session在60分钟都没等待事件，一直跑在CPU上，那么：

DB CPU= 2 \* 60 mins  ， DB Time = 2\* 60 + 0 + 0 =120

AAS = 120/60=2  正好等于OS load 2。

如果有3个session都100%仅消耗CPU，那么总有一个要wait on queue

DB CPU = 2\* 60 mins  ，wait on CPU queue= 60 mins

AAS= (120+ 60)/60=3 主机load 亦为3，此时vmstat 看waiting for run time

真实世界中？  DB Cpu = xx mins ， Non-Idle Wait= enq:TX + cursor pin S on X + latch : xxx + db file sequential read + ……….. 阿猫阿狗

## 1-1  内存参数大小

|  |
| --- |
| Cache Sizes Begin End  ~~~~~~~~~~~ ---------- ----------  Buffer Cache: 49,152M 49,152M Std Block Size: 8K  Shared Pool Size: 13,312M 13,312M Log Buffer: 334,848K |

内存管理方式：MSMM、ASMM(sga\_target)、AMM(memory\_target)

小内存有小内存的问题， 大内存有大内存的麻烦！ ORA-04031???!!

Buffer cache和shared pool size的 begin/end值在ASMM、AMM和11gR2 MSMM下可是会动的哦！

这里说 shared pool一直收缩，则在shrink过程中一些row cache 对象被lock住可能导致前台row cache lock等解析等待，最好别让shared pool shrink。如果这里shared pool一直在grow，那说明shared pool原有大小不足以满足需求(可能是大量硬解析)，结合下文的解析信息和SGA breakdown来一起诊断问题。

## 1-2   Load Profile

|  |
| --- |
| Load Profile Per Second Per Transaction Per Exec Per Call  ~~~~~~~~~~~~ --------------- --------------- ---------- ----------  DB Time(s): 256.6 0.2 0.07 0.03  DB CPU(s): 3.7 0.0 0.00 0.00  Redo size: 1,020,943.0 826.5  Logical reads: 196,888.0 159.4  Block changes: 6,339.4 5.1  Physical reads: 5,076.7 4.1  Physical writes: 379.2 0.3  User calls: 10,157.4 8.2  Parses: 204.0 0.2  Hard parses: 0.9 0.0  W/A MB processed: 5.0 0.0  Logons: 1.7 0.0  Executes: 3,936.6 3.2  Rollbacks: 1,126.3 0.9  Transactions: 1,235.3  % Blocks changed per Read: 53.49 Recursive Call %: 98.04  Rollback per transaction %: 36.57 Rows per Sort: 73.70 |

|  |  |
| --- | --- |
| 指标 | 指标含义 |
| redo size | 单位 bytes，redo size可以用来估量update/insert/delete的频率，大的redo size往往对lgwr写日志，和arch归档造成I/O压力， Per Transaction可以用来分辨是  大量小事务， 还是少量大事务。如上例每秒redo 约1MB ，每个事务800 字节，符合OLTP特征 |
| Logical Read | 单位  次数\*块数， 相当于 “人\*次”， 如上例  196,888 \* db\_block\_size=1538MB/s ， 逻辑读耗CPU，主频和CPU核数都很重要，逻辑读高则DB CPU往往高，也往往可以看到latch: cache buffer chains等待。  大量OLTP系统(例如siebel)可以高达几十乃至上百Gbytes。 |
| Block changes | 单位 次数\*块数 ， 描绘数据变化频率 |
| Physical Read | 单位次数\*块数， 如上例 5076 \* 8k = 39MB/s， 物理读消耗IO读，体现在IOPS和吞吐量等不同纬度上；但减少物理读可能意味着消耗更多CPU。好的存储 每秒物理读能力达到几GB，例如Exadata。  这个physical read包含了physical reads cache和physical reads direct |
| Physical writes | 单位  次数\*块数，主要是DBWR写datafile，也有direct path write。 dbwr长期写出慢会导致定期log file switch(checkpoint no complete) 检查点无法完成的前台等待。  这个physical write 包含了physical writes direct +physical writes from cache |
| User Calls | 单位次数，用户调用数，more details from internal |
| Parses | 解析次数，包括软解析+硬解析，软解析优化得不好，则夸张地说几乎等于每秒SQL执行次数。 即执行解析比1:1，而我们希望的是 解析一次 到处运行哦！ |
| Hard Parses | 万恶之源．　Cursor pin s on X， library cache: mutex X ， latch: row cache objects /shared pool……………..。 硬解析最好少于每秒20次 |
| W/A MB processed | 单位MB  W/A workarea  workarea中处理的数据数量 结合 In-memory Sort%， sorts (disk) PGA Aggr一起看 |
| Logons | 登陆次数， logon storm 登陆风暴，结合AUDIT审计数据一起看。短连接的附带效应是游标缓存无用 |
| Executes | 执行次数，反应执行频率 |
| Rollback | 回滚次数， 反应回滚频率， 但是这个指标不太精确，参考而已，别太当真 |
| Transactions | 每秒事务数，是数据库层的TPS，可以看做压力测试或比对性能时的一个指标，孤立看无意义 |
| % Blocks changed per Read | 每次逻辑读导致数据块变化的比率；如果’redo size’, ‘block changes’ ‘pct of blocks changed per read’三个指标都很高，则说明系统正执行大量insert/update/delete; pct of blocks changed per read =  (block changes ) /( logical reads) |
| Recursive Call % | 递归调用的比率;Recursive Call % = (recursive calls)/(user calls) |
| Rollback per transaction % | 事务回滚比率。  Rollback per transaction %= (rollback)/(transactions) |
| Rows per Sort | 平均每次排序涉及到的行数 ;  Rows per Sort= ( sorts(rows) ) / ( sorts(disk) + sorts(memory)) |

注意这些Load Profile 负载指标 在本环节提供了 2个维度 per second 和 per transaction。

* per Second:   主要是把 快照内的delta值除以 快站时间的秒数 ， 例如 在 A快照中V$SYSSTAT视图反应 table scans (long tables) 这个指标是 100 ，在B快照中V$SYSSTAT视图反应 table scans (long tables) 这个指标是 3700, 而A快照和B快照 之间 间隔了一个小时 3600秒，  则  对于  table scans (long tables) per second  就是 (  3700- 100) /3600=1。

pert Second是我们审视数据的主要维度 ，任何性能数据脱离了 时间模型则毫无意义。

在statspack/AWR出现之前的调优 洪荒时代，有很多DBA依赖V$SYSSTAT等视图中的累计 统计信息来调优，以当前的调优眼光来看，那无异于刀耕火种。

* per transaction :  基于事务的维度，与per second相比 是把除数从时间的秒数改为了该段时间内的事务数。 这个维度的很大用户是用来 识别应用特性的变化 ，若2个AWR性能报告中该维度指标 出现了大幅变化，例如 redo size从本来per transaction  1k变化为  10k per transaction，则说明SQL业务逻辑肯定发生了某些变化。

注意AWR中的这些指标 并不仅仅用来孤立地了解 Oracle数据库负载情况，实施调优工作。对于故障诊断例如HANG、Crash等，完全可以通过对比问题时段的性能报告和常规时间来对比，通过各项指标的对比往往可以找出病灶所在。

|  |
| --- |
| SELECT VALUE FROM DBA\_HIST\_SYSSTAT WHERE SNAP\_ID = :B4 AND DBID = :B3 AND INSTANCE\_NUMBER = :B2 AND STAT\_NAME  in ( "db block changes","user calls","user rollbacks","user commits",redo size","physical reads direct","physical writes","parse count (hard)","parse count (total)","session logical reads","recursive calls","redo log space requests","redo entries","sorts (memory)","sorts (disk)","sorts (rows)","logons cumulative","parse time cpu","parse time elapsed","execute count","logons current","opened cursors current","DBWR fusion writes","gcs messages sent","ges messages sent","global enqueue gets sync","global enqueue get time","gc cr blocks received","gc cr block receive time","gc current blocks received","gc current block receive time","gc cr blocks served","gc cr block build time","gc cr block flush time","gc cr block send time","gc current blocks served","gc current block pin time","gc current block flush time","gc current block send time","physical reads","physical reads direct (lob)",  SELECT TOTAL\_WAITS FROM DBA\_HIST\_SYSTEM\_EVENT WHERE SNAP\_ID = :B4 AND DBID = :B3 AND INSTANCE\_NUMBER = :B2 AND EVENT\_NAME in ("gc buffer busy","buffer busy waits"  SELECT VALUE FROM DBA\_HIST\_SYS\_TIME\_MODEL WHERE DBID = :B4 AND SNAP\_ID = :B3 AND INSTANCE\_NUMBER = :B2 AND STAT\_NAME  in  ("DB CPU","sql execute elapsed time","DB time"  SELECT VALUE FROM DBA\_HIST\_PARAMETER WHERE SNAP\_ID = :B4 AND DBID = :B3 AND INSTANCE\_NUMBER = :B2 AND PARAMETER\_NAME  in ("\_\_db\_cache\_size","\_\_shared\_pool\_size","sga\_target","pga\_aggregate\_target","undo\_management","db\_block\_size","log\_buffer","timed\_statistics","statistics\_level"  SELECT BYTES FROM DBA\_HIST\_SGASTAT WHERE SNAP\_ID = :B4 AND DBID = :B3 AND INSTANCE\_NUMBER = :B2 AND POOL IN ('shared pool', 'all pools') AND NAME  in ("free memory",  SELECT BYTES FROM DBA\_HIST\_SGASTAT WHERE SNAP\_ID = :B4 AND DBID = :B3 AND INSTANCE\_NUMBER = :B2 AND NAME = :B1 AND POOL IS NULL  SELECT (E.BYTES\_PROCESSED - B.BYTES\_PROCESSED) FROM DBA\_HIST\_PGA\_TARGET\_ADVICE B, DBA\_HIST\_PGA\_TARGET\_ADVICE E WHERE B.DBID = :B4 AND B.SNAP\_ID = :B3 AND B.INSTANCE\_NUM  BER = :B2 AND B.ADVICE\_STATUS = 'ON' AND E.DBID = B.DBID AND E.SNAP\_ID = :B1 AND E.INSTANCE\_NUMBER = B.INSTANCE\_NUMBER AND E.PGA\_TARGET\_FACTOR = 1 AND B.PGA\_TARGET\_FACT  OR = 1 AND E.ADVICE\_STATUS = 'ON'  SELECT SUM(E.TOTAL\_WAITS - NVL(B.TOTAL\_WAITS, 0)) FROM DBA\_HIST\_SYSTEM\_EVENT B, DBA\_HIST\_SYSTEM\_EVENT E WHERE B.SNAP\_ID(+) = :B4 AND E.SNAP\_ID = :B3 AND B.DBID(+) = :B2  AND E.DBID = :B2 AND B.INSTANCE\_NUMBER(+) = :B1 AND E.INSTANCE\_NUMBER = :B1 AND B.EVENT\_ID(+) = E.EVENT\_ID AND (E.EVENT\_NAME = 'latch free' OR E.EVENT\_NAME LIKE 'latch:%')  SELECT DECODE(B.TOTAL\_SQL, 0, 0, 100\*(1-B.SINGLE\_USE\_SQL/B.TOTAL\_SQL)), DECODE(E.TOTAL\_SQL, 0, 0, 100\*(1-E.SINGLE\_USE\_SQL/E.TOTAL\_SQL)), DECODE(B.TOTAL\_SQL\_MEM, 0, 0, 1  00\*(1-B.SINGLE\_USE\_SQL\_MEM/B.TOTAL\_SQL\_MEM)), DECODE(E.TOTAL\_SQL\_MEM, 0, 0, 100\*(1-E.SINGLE\_USE\_SQL\_MEM/E.TOTAL\_SQL\_MEM)) FROM DBA\_HIST\_SQL\_SUMMARY B, DBA\_HIST\_SQL\_SUMM  ARY E WHERE B.SNAP\_ID = :B4 AND E.SNAP\_ID = :B3 AND B.INSTANCE\_NUMBER = :B2 AND E.INSTANCE\_NUMBER = :B2 AND B.DBID = :B1 AND E.DBID = :B1  SELECT EVENT, WAITS, TIME, DECODE(WAITS, NULL, TO\_NUMBER(NULL), 0, TO\_NUMBER(NULL), TIME/WAITS\*1000) AVGWT, PCTWTT, WAIT\_CLASS FROM (SELECT EVENT, WAITS, TIME, PCTWTT,  WAIT\_CLASS FROM (SELECT E.EVENT\_NAME EVENT, E.TOTAL\_WAITS - NVL(B.TOTAL\_WAITS,0) WAITS, (E.TIME\_WAITED\_MICRO - NVL(B.TIME\_WAITED\_MICRO,0)) / 1000000 TIME, 100 \* (E.TIME  \_WAITED\_MICRO - NVL(B.TIME\_WAITED\_MICRO,0)) / :B1 PCTWTT, E.WAIT\_CLASS WAIT\_CLASS FROM DBA\_HIST\_SYSTEM\_EVENT B, DBA\_HIST\_SYSTEM\_EVENT E WHERE B.SNAP\_ID(+) = :B5 AND E.S  NAP\_ID = :B4 AND B.DBID(+) = :B3 AND E.DBID = :B3 AND B.INSTANCE\_NUMBER(+) = :B2 AND E.INSTANCE\_NUMBER = :B2 AND B.EVENT\_ID(+) = E.EVENT\_ID AND E.TOTAL\_WAITS > NVL(B.TO  TAL\_WAITS,0) AND E.WAIT\_CLASS != 'Idle' UNION ALL SELECT 'CPU time' EVENT, TO\_NUMBER(NULL) WAITS, :B6 /1000000 TIME, 100 \* :B6 / :B1 PCTWTT, NULL WAIT\_CLASS FROM DUAL W  HERE :B6 > 0) ORDER BY TIME DESC, WAITS DESC) WHERE ROWNUM <= :B7  SELECT SUM(E.TIME\_WAITED\_MICRO - NVL(B.TIME\_WAITED\_MICRO,0)) FROM DBA\_HIST\_SYSTEM\_EVENT B, DBA\_HIST\_SYSTEM\_EVENT E WHERE B.SNAP\_ID(+) = :B4 AND E.SNAP\_ID = :B3 AND B.DB  ID(+) = :B2 AND E.DBID = :B2 AND B.INSTANCE\_NUMBER(+) = :B1 AND E.INSTANCE\_NUMBER = :B1 AND B.EVENT\_ID(+) = E.EVENT\_ID AND E.WAIT\_CLASS = 'User I/O'  SELECT (E.ESTD\_LC\_TIME\_SAVED - B.ESTD\_LC\_TIME\_SAVED) FROM DBA\_HIST\_SHARED\_POOL\_ADVICE B, DBA\_HIST\_SHARED\_POOL\_ADVICE E WHERE B.DBID = :B3 AND B.INSTANCE\_NUMBER = :B2 AN  D B.SNAP\_ID = :B4 AND E.DBID = :B3 AND E.INSTANCE\_NUMBER = :B2 AND E.SNAP\_ID = :B1 AND E.SHARED\_POOL\_SIZE\_FACTOR = 1 AND B.SHARED\_POOL\_SIZE\_FACTOR = 1 |

## 1-3  Instance Efficiency Percentages (Target 100%)

|  |
| --- |
| Instance Efficiency Percentages (Target 100%)  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~  Buffer Nowait %: 99.97 Redo NoWait %: 100.00  Buffer Hit %: 97.43 In-memory Sort %: 100.00  Library Hit %: 99.88 Soft Parse %: 99.58  Execute to Parse %: 94.82 Latch Hit %: 99.95  Parse CPU to Parse Elapsd %: 1.75 % Non-Parse CPU: 99.85 |

上述所有指标 的目标均为100%，即越大越好，在少数bug情况下可能超过100%或者为负值。

* 80%以上  %Non-Parse CPU
* 90%以上  Buffer Hit%, In-memory Sort%, Soft Parse%
* 95%以上  Library Hit%, Redo Nowait%, Buffer Nowait%
* 98%以上  Latch Hit%

### Buffer Nowait %

Buffer Nowait %  session申请一个buffer(兼容模式)不等待的次数比例。 需要访问buffer时立即可以访问的比率，  不兼容的情况 在9i中是 buffer busy waits，从10g以后 buffer busy waits 分离为 buffer busy wait 和 read by other session2个等待事件 :

9i 中 waitstat的总次数基本等于buffer busy waits等待事件的次数

|  |
| --- |
| SQL> select sum(TOTAL\_WAITS) from v$system\_event where event='buffer busy waits';  SUM(TOTAL\_WAITS)  —————-  33070394  SQL> select sum(count) from v$waitstat;  SUM(COUNT)  ———-  33069335 |

10g waitstat的总次数基本等于 buffer busy waits 和 read by other session 等待的次数总和

|  |
| --- |
| SQL> select sum(TOTAL\_WAITS) from v$system\_event where event='buffer busy waits' or event='read by other session';  SUM(TOTAL\_WAITS)  —————-  60675815  SQL> select sum(count) from v$waitstat;  SUM(COUNT)  ———-  60423739 |

Buffer Nowait %的计算公式是 sum(v$waitstat.wait\_count) / (v$sysstat statistic session logical reads)，例如在AWR中：

|  |  |  |  |
| --- | --- | --- | --- |
| Class | Waits | Total Wait Time (s) | Avg Time (ms) |
| data block | 24,543 | 2,267 | 92 |
| undo header | 743 | 2 | 3 |
| undo block | 1,116 | 0 | 0 |
| 1st level bmb | 35 | 0 | 0 |

|  |  |  |  |
| --- | --- | --- | --- |
| session logical reads | 40,769,800 | 22,544.84 | 204.71 |

|  |  |
| --- | --- |
| Buffer Nowait %: | 99.94 |

Buffer Nowait= (  40,769,800 – (24543+743+1116+35))/ ( 40,769,800) = 0.99935= 99.94%

SELECT SUM(WAIT\_COUNT) FROM DBA\_HIST\_WAITSTAT WHERE SNAP\_ID = :B3 AND DBID = :B2 AND INSTANCE\_NUMBER = :B1

### buffer HIT%:

2、buffer HIT%: 经典的经典，高速缓存命中率，反应物理读和缓存命中间的纠结，但这个指标即便99% 也不能说明物理读等待少了

不合理的db\_cache\_size，或者是SGA自动管理ASMM /Memory 自动管理AMM下都可能因为db\_cache\_size过小引起大量的db file sequential /scattered read等待事件； maclean曾经遇到过因为大量硬解析导致ASMM 下shared pool共享池大幅度膨胀，而db cache相应缩小shrink的例子，最终db cache收缩到只有几百兆，本来没有的物理读等待事件都大幅涌现出来 。

此外与 buffer HIT%相关的指标值得关注的还有 table scans(long tables) 大表扫描这个统计项目、此外相关的栏目还有Buffer Pool Statistics 、Buffer Pool Advisory等（如果不知道在哪里，直接找一个AWR 去搜索这些关键词即可)。

buffer HIT%在 不同版本有多个计算公式：

在9i中

Buffer Hit Ratio = 1 – ((physical reads – physical reads direct – physical reads direct (lob)) / (db block gets + consistent gets – physical reads direct – physical reads direct (lob))

在10g以后：

Buffer Hit Ratio=  1 – ((‘physical reads cache’) / (‘consistent gets from cache’ + ‘db block gets from cache’)

注意：但是实际AWR中 似乎还是按照9i中的算法，虽然算法的区别对最后算得的比率影响不大。

对于buffer hit % 看它的命中率有多高没有意义，主要是关注 未命中的次数有多少。通过上述公式很容易反推出未命中的物理读的次数。

db block gets 、consistent gets 以及 session logical reads的关系如下：

db block gets＝db block gets direct＋　db block gets from cache

consistent gets　＝　consistent gets from cache＋　consistent gets direct

consistent gets from cache＝　consistent gets – examination  + else

consistent gets – examination==>指的是不需要pin buffer直接可以执行consistent get的次数，常用于索引，只需要一次latch get

session logical reads = db block gets +consistent gets

其中physical reads 、physical reads cache、physical reads direct、physical reads direct (lob)几者的关系为：

physical reads = physical reads cache +　physical reads direct

这个公式其实说明了 物理读有2种 ：

物理读进入buffer cache中 ，是常见的模式 physical reads cache

物理读直接进入PGA 直接路径读， 即physical reads direct

|  |  |  |
| --- | --- | --- |
| physical reads | 8 | Total number of data blocks read from disk. This value can be greater than the value of “physical reads direct” plus “physical reads cache” as reads into process private buffers also included in this statistic. |

|  |  |  |
| --- | --- | --- |
| physical reads cache | 8 | Total number of data blocks read from disk into the buffer cache. This is a subset of “physical reads” statistic. |

|  |  |  |
| --- | --- | --- |
| physical reads direct | 8 | Number of reads directly from disk, bypassing the buffer cache. For example, in high bandwidth, data-intensive operations such as parallel query, reads of disk blocks bypass the buffer cache to maximize transfer rates and to prevent the premature aging of shared data blocks resident in the buffer cache. |

physical reads direct = physical reads direct (lob) + physical reads direct temporary tablespace +  physical reads direct(普通)

这个公式也说明了 直接路径读 分成三个部分：

physical reads direct (lob) 直接路径读LOB对象

physical reads direct temporary tablespace  直接路径读临时表空间

physical read direct(普通)   普通的直接路径读， 一般是11g开始的自动的大表direct path read和并行引起的direct path read

physical writes direct= physical writes direct (lob)+ physical writes direct temporary tablespace

DBWR checkpoint buffers written = DBWR thread checkpoint buffers written+ DBWR tablespace checkpoint buffers written+ DBWR PQ tablespace checkpoint buffers written+….

### Redo nowait%:

3、Redo nowait%: session在生成redo entry时不用等待的比例，redo相关的资源争用例如redo space request争用可能造成生成redo时需求等待。此项数据来源于v$sysstat中的(redo log space requests/redo entries)。 一般来说10g以后不太用关注log\_buffer参数的大小，需要关注是否有十分频繁的 log switch ； 过小的redo logfile size 如果配合较大的SGA和频繁的commit提交都可能造成该问题。 考虑增到redo logfile 的尺寸 : 1~4G 每个，7~10组都是合适的。同时考虑优化redo logfile和datafile 的I/O。

### In-memory Sort%:

4、In-memory Sort%:这个指标因为它不计算workarea中所有的操作类型，所以现在越来越鸡肋了。 纯粹在内存中完成的排序比例。数据来源于v$sysstat statistics sorts (disk) 和 sorts (memory)，  In-memory Sort% =  sort(memory) / ( sort(disk)+ sort(memory) )

### Library Hit%:

5、

Library Hit%:  library cache命中率，申请一个library cache object例如一个SQL cursor时，其已经在library cache中的比例。 数据来源  V$librarycache的pins和pinhits。 合理值：>95%       ，该比例来源于1- ( Σ(pin Requests \* Pct Miss) / Sum(Pin Requests) )

[维护这个指标的重点是 保持shared pool共享池有足够的Free Memory，且没有过多的内存碎片，具体可以参考这里。  显然过小的shared pool可用空间会导致library cache object被aged out换出共享池。](http://www.askmaclean.com/archives/ora-4031-common-analysisdiagnostic-scripts.html)

此外保证SQL语句绑定变量和游标可以共享也是很重要的因素。

|  |
| --- |
| Library Cache Activity DB/Inst: G10R25/G10R25 Snaps: 2964-2965  -> "Pct Misses" should be very low http://www.askmaclean.com  Get Pct Pin Pct Invali-  Namespace Requests Miss Requests Miss Reloads dations  --------------- ------------ ------ -------------- ------ ---------- --------  BODY 5 0.0 6 16.7 1 0  CLUSTER 10 0.0 26 0.0 0 0  SQL AREA 601,357 99.8 902,828 99.7 47 2  TABLE/PROCEDURE 83 9.6 601,443 0.0 48 0 |

|  |  |  |
| --- | --- | --- |
| GETS | NUMBER | Number of times a lock was requested for objects of this namespace |
| GETHITS | NUMBER | Number of times an object’s handle was found in memory |
| GETHITRATIO | NUMBER | Ratio of GETHITS to GETS |
| PINS | NUMBER | Number of times a PIN was requested for objects of this namespace |
| PINHITS | NUMBER | Number of times all of the metadata pieces of the library object were found in memory |
| PINHITRATIO | NUMBER | Ratio of PINHITS to PINS |
| RELOADS | NUMBER | Any PIN of an object that is not the first PIN performed since the object handle was created, and which requires loading the object from disk |
| INVALIDATIONS | NUMBER | Total number of times objects in this namespace were marked invalid because a dependent object was modified |

SELECT SUM(PINS), SUM(PINHITS) FROM DBA\_HIST\_LIBRARYCACHE WHERE SNAP\_ID = :B3 AND DBID = :B2 AND INSTANCE\_NUMBER = :B1

### Soft Parse:

6、

Soft Parse: 软解析比例，无需多说的经典指标，数据来源v$sysstat statistics的parse count(total)和parse count(hard)。 合理值>95%

Soft Parse %是AWR中另一个重要的解析指标，该指标反应了快照时间内 软解析次数 和 总解析次数 (soft+hard 软解析次数+硬解析次数)的比值，若该指标很低，那么说明了可能存在剧烈的hard parse硬解析，大量的硬解析会消耗更多的CPU时间片并产生解析争用(此时可以考虑使用cursor\_sharing=FORCE)； 理论上我们总是希望 Soft Parse % 接近于100%， 但并不是说100%的软解析就是最理想的解析状态，[通过设置 session\_cached\_cursors参数](http://www.askmaclean.com/archives/script-to-tune-the-session_cached_cursors-and-open_cursors-parameters.html)和反复重用游标我们可以让解析来的更轻量级，即通俗所说的利用会话缓存游标实现的软软解析(soft soft parse)。

### Execute  to Parse%

7、

Execute  to Parse% 指标反映了执行解析比 其公式为 1-(parse/execute) , 目标为100% 及接近于只 执行而不解析。 数据来源v$sysstat statistics parse count (total) 和execute count

在oracle中解析往往是执行的先提工作，但是通过游标共享 可以解析一次 执行多次， 执行解析可能分成多种场景：

hard coding => 硬编码代码 硬解析一次 ，执行一次， 则理论上其执行解析比 为 1:1 ，则理论上Execute to Parse =0 极差，且soft parse比例也为0%

绑定变量但是仍软解析=》 软解析一次，执行一次 ， 这种情况虽然比前一种好 但是执行解析比(这里的parse，包含了软解析和硬解析)仍是1:1， 理论上Execute to Parse =0 极差， 但是soft parse比例可能很高

使用 静态SQL、动态绑定、session\_cached\_cursor、open cursors等技术实现的 解析一次，执行多次， 执行解析比为N:1， 则 Execute to Parse= 1- (1/N) 执行次数越多 Execute to Parse越接近100% ，这种是我们在OLTP环境中喜闻乐见的！

通俗地说 soft parse% 反映了软解析率， 而软解析在oracle中仍是较昂贵的操作， 我们希望的是解析1次执行N次，如果每次执行均需要软解析，那么虽然soft parse%=100% 但是parse time仍可能是消耗DB TIME的大头。

Execute to Parse反映了 执行解析比，Execute to Parse和soft parse% 都很低 那么说明确实没有绑定变量 ， 而如果 soft parse% 接近99% 而Execute to Parse 不足90% 则说明没有执行解析比低， 需要通过 静态SQL、动态绑定、session\_cached\_cursor、open cursors等技术减少软解析。

### Latch Hit%

8、

Latch Hit%: willing-to-wait latch闩申请不要等待的比例。 数据来源V$latch gets和misses

|  |
| --- |
| Latch Name  ----------------------------------------    Get Requests      Misses      Sleeps  Spin Gets   Sleep1   Sleep2   Sleep3  -------------- ----------- ----------- ---------- -------- -------- --------  shared pool       9,988,637         364          23        341        0        0        0  library cache       6,753,468         152           6        146        0        0        0  Memory Management Latch             369           1           1          0        0        0        0  qmn task queue latch              24           1           1          0        0        0        0 |

Latch Hit%:=  (1 – (Sum(misses) / Sum(gets)))

关于Latch的更多信息内容可以参考 AWR后面的专栏Latch Statistics， 注意对于一个并发设计良好的OLTP应用来说，Latch、Enqueue等并发控制不应当成为系统的主要瓶颈， 同时对于这些并发争用而言 堆积硬件CPU和内存 很难有效改善性能。

SELECT SUM(GETS), SUM(MISSES) FROM DBA\_HIST\_LATCH WHERE SNAP\_ID = :B3 AND DBID = :B2 AND INSTANCE\_NUMBER = :B1

### Parse CPU To Parse Elapsd:

9、

Parse CPU To Parse Elapsd:该指标反映了 快照内解析CPU时间和总的解析时间的比值(Parse CPU Time/ Parse Elapsed Time)； 若该指标水平很低，那么说明在整个解析过程中 实际在CPU上运算的时间是很短的，而主要的解析时间都耗费在各种其他非空闲的等待事件上了(如latch:shared pool,row cache lock之类等)   数据来源 V$sysstat 的 parse time cpu和parse time elapsed

### %Non-Parse CPU

10、

%Non-Parse CPU 非解析cpu比例，公式为  (DB CPU – Parse CPU)/DB CPU，  若大多数CPU都用在解析上了，则可能好钢没用在刃上了。 数据来源 v$sysstat 的 parse time cpu和 cpu used by this session

## 1-4    Shared Pool Statistics

|  |
| --- |
| Shared Pool Statistics Begin End  ------ ------  Memory Usage %: 84.64 79.67  % SQL with executions>1: 93.77 24.69  % Memory for SQL w/exec>1: 85.36 34.8 |

该环节提供一个大致的SQL重用及shared pool内存使用的评估。 应用是否共享SQL? 有多少内存是给只运行一次的SQL占掉的，对比共享SQL呢？

如果该环节中% SQL with executions>1的 比例 小于%90 ， 考虑用下面链接的SQL去抓 硬编码的非绑定变量SQL语句。

[利用FORCE\_MATCHING\_SIGNATURE捕获非绑定变量SQL](http://www.askmaclean.com/archives/%E5%88%A9%E7%94%A8force_matching_signature%E6%8D%95%E8%8E%B7%E9%9D%9E%E7%BB%91%E5%AE%9A%E5%8F%98%E9%87%8Fsql.html)

Memory Usage %:    (shared pool 的实时大小- shared pool free memory)/ shared pool 的实时大小， 代表shared pool的空间使用率，虽然有使用率但没有标明碎片程度

% SQL with executions>1      复用的SQL占总的SQL语句的比率,数据来源 DBA\_HIST\_SQL\_SUMMARY 的 SINGLE\_USE\_SQL和TOTAL\_SQL：1 – SINGLE\_USE\_SQL / TOTAL\_SQL

% Memory for SQL w/exec>1   执行2次以上的SQL所占内存占总的SQL内存的比率，数据来源DBA\_HIST\_SQL\_SUMMARY 的SINGLE\_USE\_SQL\_MEM和TOTAL\_SQL\_MEM：1 – SINGLE\_USE\_SQL\_MEM / TOTAL\_SQL\_MEM

==》上面2个指标也可以用来大致了解shared pool中的内存碎片程序，因为SINGLE\_USE\_SQL 单次执行的SQL多的话，那么显然可能有较多的共享池内存碎片

SQL复用率低的原因一般来说就是硬绑定变量(hard Coding)未合理使用绑定变量(bind variable)，对于这种现象短期无法修改代表使用绑定变量的可以ALTER SYSTEM SET CURSOR\_SHARING=FORCE; 来绕过问题，对于长期来看还是要修改代码绑定变量。   Oracle 从11g开始宣称今后将废弃CURSOR\_SHARING的SIMILAR选项，同时SIMILAR选项本身也造成了很多问题，所以一律不推荐用CURSOR\_SHARING=SIMILAR。

如果memory usage%比率一直很高，则可以关注下后面sga breakdown中的shared pool free memory大小，一般推荐至少让free memroy有个300~500MB 以避免隐患。

## 1-5 Top 5 Timed Events

|  |
| --- |
| Top 5 Timed Events Avg %Total  ~~~~~~~~~~~~~~~~~~ wait Call  Event Waits Time (s) (ms) Time Wait Class  ------------------------------ ------------ ----------- ------ ------ ----------  gc buffer busy 79,083 73,024 923 65.4 Cluster  enq: TX - row lock contention 35,068 17,123 488 15.3 Applicatio  CPU time 12,205 10.9  gc current request 2,714 3,315 1221 3.0 Cluster  gc cr multi block request 83,666 1,008 12 0.9 Cluster |

基于Wait Interface的调优是目前的主流！每个指标都重要！

基于命中比例的调优，好比是统计局的报告， 张财主家财产100万，李木匠家财产1万， 平均财产50.5万。

基于等待事件的调优，好比马路上100辆汽车的行驶记录表，上车用了几分钟， 红灯等了几分钟，拥堵塞了几分钟。。。

丰富的等待事件以足够的细节来描绘系统运行的性能瓶颈，这是Mysql梦寐以求的东西……

Waits : 该等待事件发生的次数， 对于DB CPU此项不可用

Times : 该等待事件消耗的总计时间，单位为秒， 对于DB CPU 而言是前台进程所消耗CPU时间片的总和，但不包括Wait on CPU QUEUE

Avg Wait(ms)  :  该等待事件平均等待的时间， 实际就是  Times/Waits，单位ms， 对于DB CPU此项不可用

% Total Call Time， 该等待事件占总的call time的比率

total call time  =  total CPU time + total wait time for non-idle events

% Total Call Time  =  time for each timed event / total call time

Wait Class: 等待类型：

Concurrency,System I/O,User I/O,Administrative,Other,Configuration,Scheduler,Cluster,Application,Idle,Network,Commit

### CPU 上在干什么？

逻辑读？ 解析？Latch spin? PL/SQL、函数运算?

DB CPU/CPU time是Top 1 是好事情吗？  未必！

注意DB CPU不包含 wait on cpu queue！

|  |
| --- |
| SELECT e.event\_name event,  e.total\_waits - NVL (b.total\_waits, 0) waits,  DECODE (  e.total\_waits - NVL (b.total\_waits, 0),  0, TO\_NUMBER (NULL),  DECODE (  e.total\_timeouts - NVL (b.total\_timeouts, 0),  0, TO\_NUMBER (NULL),  100  \* (e.total\_timeouts - NVL (b.total\_timeouts, 0))  / (e.total\_waits - NVL (b.total\_waits, 0))))  pctto,  (e.time\_waited\_micro - NVL (b.time\_waited\_micro, 0)) / 1000000 time,  DECODE (  (e.total\_waits - NVL (b.total\_waits, 0)),  0, TO\_NUMBER (NULL),  ( (e.time\_waited\_micro - NVL (b.time\_waited\_micro, 0)) / 1000)  / (e.total\_waits - NVL (b.total\_waits, 0)))  avgwt,  DECODE (e.wait\_class, 'Idle', 99, 0) idle  FROM dba\_hist\_system\_event b, dba\_hist\_system\_event e  WHERE b.snap\_id(+) = &bid  AND e.snap\_id = &eid  --AND b.dbid(+) = :dbid  --AND e.dbid = :dbid  AND b.instance\_number(+) = 1  AND e.instance\_number = 1  AND b.event\_id(+) = e.event\_id  AND e.total\_waits > NVL (b.total\_waits, 0)  AND e.event\_name NOT IN  ('smon timer',  'pmon timer',  'dispatcher timer',  'dispatcher listen timer',  'rdbms ipc message')  ORDER BY idle,  time DESC,  waits DESC,  event |

# 几种常见的等待事件

=========================>

[db file scattered read,  Avg wait time应当小于20ms  如果数据库执行全表扫描或者是全索引扫描会执行 Multi block I/O ，此时等待物理I/O 结束会出现此等待事件。一般会从应用程序（SQL），I/O 方面入手调整; 注意和《Instance Activity Stats》中的index fast full scans (full) 以及 table scans (long tables)集合起来一起看。](http://www.askmaclean.com/archives/db-file-scattered-read.html)

db file sequential read ，该等待事件Avg wait time平均单次等待时间应当小于20ms

”db file sequential read”单块读等待是一种最为常见的物理IO等待事件，这里的sequential指的是将数据块读入到相连的内存空间中(contiguous memory space)，而不是指所读取的数据块是连续的。该wait event可能在以下情景中发生:

<http://www.askmaclean.com/archives/db-file-sequential-read-wait-event.html>

[latch free　　其实是未获得latch ，而进入latch sleep，见《全面解析9i以后Oracle Latch闩锁原理》](http://www.askmaclean.com/archives/oracle-latch.html)

enq:XX           队列锁等待，视乎不同的队列锁有不同的情况：

[你有多了解Oracle Enqueue lock队列锁机制？](http://www.askmaclean.com/archives/oracle-enqueue-lock.html)

[Oracle队列锁: Enqueue HW](http://www.askmaclean.com/archives/enqueue-lock-high-water-mark.html)

[Oracle队列锁enq:US,Undo Segment](http://www.askmaclean.com/archives/enqueue-lock-us-undo-segment.html)

[enq: TX – row lock/index contention、allocate ITL等待事件](http://www.askmaclean.com/archives/enqueue-tx-row-lock-index-itl-wait-event.html)

[enq: TT – contention等待事件](http://www.askmaclean.com/archives/enq-tt-contention.html)

[Oracle队列锁enq:TS,Temporary Segment (also TableSpace)](http://www.askmaclean.com/archives/enq-ts-temporary-segment.html)

[enq: JI – contention等待事件](http://www.askmaclean.com/archives/enq-ji-contention.html)

[enq: US – contention等待事件](http://www.askmaclean.com/archives/enq-us-contention.html)

[enq: TM – contention等待事件](http://www.askmaclean.com/archives/enq-tm-contention.html)

[enq: RO fast object reuse等待事件](http://www.askmaclean.com/archives/enq-ro-fast-object-reuse.html)

[enq: HW – contention等待事件](http://www.askmaclean.com/archives/enqueue-lock-us-undo-segment.html)

free buffer waits：是由于无法找到可用的buffer cache 空闲区域，需要等待DBWR 写入完成引起

一般是由于

低效的sql

过小的buffer cache

DBWR 工作负荷过量

buffer busy wait/ read by other session  一般以上2个等待事件可以归为一起处理，建议客户都进行监控 。 以上等待时间可以由如下操作引起

select/select —- read by other session: 由于需要从 数据文件中将数据块读入 buffer cache 中引起，有可能是 大量的 逻辑/物理读  ;或者过小的 buffer cache 引起

select/update —- buffer busy waits/ read by other session  是由于更新某数据块后 需要在undo 中 重建构建 过去时间的块，有可能伴生 enq:cr-contention 是由于大量的物理读/逻辑读造成。

update/update —- buffer busy waits 由于更新同一个数据块（非同一行，同一行是enq:TX-contention） 此类问题是热点块造成

insert/insert —- buffer busy waits  是由于freelist 争用造成，可以将表空间更改为ASSM 管理 或者加大freelist 。

write complete waits :一般此类等待事件是由于 DBWR 将脏数据写入 数据文件，其他进程如果需要修改 buffer cache会引起此等待事件，一般是 I/O 性能问题或者是DBWR 工作负荷过量引起

Wait time  1 Seconds.

control file parallel write：频繁的更新控制文件会造成大量此类等待事件，如日志频繁切换，检查点经常发生，nologging 引起频繁的数据文件更改，I/O 系统性能缓慢。

log file sync：一般此类等待时间是由于 LGWR 进程讲redo log buffer 写入redo log 中发生。如果此类事件频繁发生，可以判断为：

commit 次数是否过多

I/O 系统问题

重做日志是否不必要被创建

redo log buffer 是否过大

## 2-1 Time Model Statistics

|  |
| --- |
| Time Model Statistics DB/Inst: ITSCMP/itscmp2 Snaps: 70719-70723  -> Total time in database user-calls (DB Time): 883542.2s  -> Statistics including the word "background" measure background process  time, and so do not contribute to the DB time statistic  -> Ordered by % or DB time desc, Statistic name  Statistic Name Time (s) % of DB Time  ------------------------------------------ ------------------ ------------  sql execute elapsed time 805,159.7 91.1  sequence load elapsed time 41,159.2 4.7  DB CPU 20,649.1 2.3  parse time elapsed 1,112.8 .1  hard parse elapsed time 995.2 .1  hard parse (sharing criteria) elapsed time 237.3 .0  hard parse (bind mismatch) elapsed time 227.6 .0  connection management call elapsed time 29.7 .0  PL/SQL execution elapsed time 9.2 .0  PL/SQL compilation elapsed time 6.6 .0  failed parse elapsed time 2.0 .0  repeated bind elapsed time 0.4 .0  DB time 883,542.2  background elapsed time 25,439.0  background cpu time 1,980.9  ------------------------------------------------------------- |

Time Model Statistics几个特别有用的时间指标：

parse time elapsed、hard parse elapsed time 结合起来看解析是否是主要矛盾，若是则重点是软解析还是硬解析

sequence load elapsed time sequence序列争用是否是问题焦点

PL/SQL compilation elapsed time PL/SQL对象编译的耗时

注意PL/SQL execution elapsed time  纯耗费在PL/SQL解释器上的时间。不包括花在执行和解析其包含SQL上的时间

connection management call elapsed time 建立数据库session连接和断开的耗时

failed parse elapsed time 解析失败，例如由于ORA-4031

hard parse (sharing criteria) elapsed time  由于无法共享游标造成的硬解析

hard parse (bind mismatch) elapsed time  由于bind type or bind size 不一致造成的硬解析

注意该时间模型中的指标存在包含关系所以Time Model Statistics加起来超过100%再正常不过

1) background elapsed time

2) background cpu time

3) RMAN cpu time (backup/restore)

1) DB time

2) DB CPU

2) connection management call elapsed time

2) sequence load elapsed time

2) sql execute elapsed time

2) parse time elapsed

3) hard parse elapsed time

4) hard parse (sharing criteria) elapsed time

5) hard parse (bind mismatch) elapsed time

3) failed parse elapsed time

4) failed parse (out of shared memory) elapsed time

2) PL/SQL execution elapsed time

2) inbound PL/SQL rpc elapsed time

2) PL/SQL compilation elapsed time

2) Java execution elapsed time

2) repeated bind elapsed time

## 2-2 Foreground Wait Class

|  |
| --- |
| Foreground Wait Class  -> s - second, ms - millisecond - 1000th of a second  -> ordered by wait time desc, waits desc  -> %Timeouts: value of 0 indicates value was < .5%. Value of null is truly 0  -> Captured Time accounts for 102.7% of Total DB time 883,542.21 (s)  -> Total FG Wait Time: 886,957.73 (s) DB CPU time: 20,649.06 (s)  Avg  %Time Total Wait wait  Wait Class Waits -outs Time (s) (ms) %DB time  -------------------- ---------------- ----- ---------------- -------- ---------  Cluster 9,825,884 1 525,134 53 59.4  Concurrency 688,375 0 113,782 165 12.9  User I/O 34,405,042 0 76,695 2 8.7  Commit 172,193 0 62,776 365 7.1  Application 11,422 0 57,760 5057 6.5  Configuration 19,418 1 48,889 2518 5.5  DB CPU 20,649 2.3  Other 1,757,896 94 924 1 0.1  System I/O 30,165 0 598 20 0.1  Network 171,955,673 0 400 0 0.0  Administrative 2 100 0 101 0.0  -------------------------------------------------------------  select distinct wait\_class from v$event\_name;  WAIT\_CLASS  ----------------------------------------------------------------  Concurrency  User I/O  System I/O  Administrative  Other  Configuration  Scheduler  Cluster  Application  Queueing  Idle  Network  Commit |

Wait Class: 等待事件的类型，如上查询所示，被分作12个类型。  10.2.0.5有916个等待事件，其中Other类型占622个。

Waits:  该类型所属等待事件在快照时间内的等待次数

%Time Out  等待超时的比率， 未 超时次数/waits  \* 100 (%)

Total Wait Time: 该类型所属等待事件总的耗时，单位为秒

Avg Wait(ms) : 该类型所属等待事件的平均单次等待时间，单位为ms ，实际这个指标对commit 和 user i/o 以及system i/o类型有点意义，其他等待类型由于等待事件差异较大所以看平均值的意义较小

waits / txn:   该类型所属等待事件的等待次数和事务比

Other 类型，遇到该类型等待事件 的话 常见的原因是Oracle Bug或者 网络、I/O存在问题， 一般推荐联系Maclean。

Concurrency 类型   并行争用类型的等待事件，  典型的如 latch: shared pool、latch: library cache、row cache lock、library cache pin/lock

Cluster 类型  为Real Application Cluster RAC环境中的等待事件， 需要注意的是 如果启用了RAC option，那么即使你的集群中只启动了一个实例，那么该实例也可能遇到 Cluster类型的等待事件, 例如gc buffer busy

System I/O  主要是后台进程维护数据库所产生的I/O，例如control file parallel write 、log file parallel write、db file parallel write。

User I/O    主要是前台进程做了一些I/O操作，并不是说后台进程不会有这些等待事件。 典型的如db file sequential/scattered  read、direct path read

Configuration  由于配置引起的等待事件，  例如 日志切换的log file switch completion (日志文件 大小/数目 不够)，sequence的enq: SQ – contention (Sequence 使用nocache) ； Oracle认为它们是由于配置不当引起的，但实际未必真是这样的配置引起的。

Application  应用造成的等待事件， 例如enq: TM – contention和enq: TX – row lock contention； Oracle认为这是由于应用设计不当造成的等待事件， 但实际这些Application class 等待可能受到 Concurrency、Cluster、System I/O 、User I/O等多种类型等待的影响，例如本来commit只要1ms ，则某一行数据仅被锁定1ms， 但由于commit变慢 从而释放行锁变慢，引发大量的enq: TX – row lock contention等待事件。

Commit  仅log file sync ，log file sync的影响十分广泛，值得我们深入讨论。

Network :  网络类型的等待事件 例如 SQL\*Net more data to client  、SQL\*Net more data to dblink

Idle 空闲等待事件 ，最为常见的是rdbms ipc message (等待实例内部的ipc通信才干活，即别人告知我有活干，我才干，否则我休息==》Idle)， SQL\*Net message from client(等待SQL\*NET传来信息，否则目前没事干)

## 2-3 前台等待事件

|  |
| --- |
| Foreground Wait Events Snaps: 70719-70723  -> s - second, ms - millisecond - 1000th of a second  -> Only events with Total Wait Time (s) >= .001 are shown  -> ordered by wait time desc, waits desc (idle events last)  -> %Timeouts: value of 0 indicates value was < .5%. Value of null is truly 0  Avg  %Time Total Wait wait Waits % DB  Event Waits -outs Time (s) (ms) /txn time  -------------------------- ------------ ----- ---------- ------- -------- ------  gc buffer busy acquire 3,274,352 3 303,088 93 13.3 34.3  gc buffer busy release 387,673 2 128,114 330 1.6 14.5  enq: TX - index contention 193,918 0 97,375 502 0.8 11.0  cell single block physical 30,738,730 0 63,606 2 124.8 7.2  log file sync 172,193 0 62,776 365 0.7 7.1  gc current block busy 146,154 0 53,027 363 0.6 6.0  enq: TM - contention 1,060 0 47,228 44555 0.0 5.3  enq: SQ - contention 17,431 0 35,683 2047 0.1 4.0  gc cr block busy 105,204 0 33,746 321 0.4 3.8  buffer busy waits 279,721 0 12,646 45 1.1 1.4  enq: HW - contention 1,201 3 12,192 10151 0.0 1.4  enq: TX - row lock content 9,231 0 10,482 1135 0.0 1.2  cell multiblock physical r 247,903 0 6,547 26 1.0 .7 |

Foreground Wait Events 前台等待事件，数据主要来源于DBA\_HIST\_SYSTEM\_EVENT

Event 等待事件名字

Waits  该等待事件在快照时间内等待的次数

%Timeouts :  每一个等待事件有其超时的设置，例如buffer busy waits 一般为3秒， Write Complete Waits的 timeout为1秒，如果等待事件 单次等待达到timeout的时间，则会进入下一次该等待事件

Total Wait Time  该等待事件 总的消耗的时间 ，单位为秒

Avg wait(ms): 该等待事件的单次平均等待时间，单位为毫秒

Waits/Txn: 该等待事件的等待次数和事务比

## 2-4 后台等待事件

|  |
| --- |
| Background Wait Events Snaps: 70719-70723  -> ordered by wait time desc, waits desc (idle events last)  -> Only events with Total Wait Time (s) >= .001 are shown  -> %Timeouts: value of 0 indicates value was < .5%. Value of null is truly 0  Avg  %Time Total Wait wait Waits % bg  Event Waits -outs Time (s) (ms) /txn time  -------------------------- ------------ ----- ---------- ------- -------- ------  db file parallel write 90,979 0 7,831 86 0.4 30.8  gcs log flush sync 4,756,076 6 4,714 1 19.3 18.5  enq: CF - contention 2,123 40 4,038 1902 0.0 15.9  control file sequential re 90,227 0 2,380 26 0.4 9.4  log file parallel write 108,383 0 1,723 16 0.4 6.8  control file parallel writ 4,812 0 988 205 0.0 3.9  Disk file operations I/O 26,216 0 731 28 0.1 2.9  flashback log file write 9,870 0 720 73 0.0 2.8  LNS wait on SENDREQ 202,747 0 600 3 0.8 2.4  ASM file metadata operatio 15,801 0 344 22 0.1 1.4  cell single block physical 39,283 0 341 9 0.2 1.3  LGWR-LNS wait on channel 183,443 18 203 1 0.7 .8  gc current block busy 122 0 132 1082 0.0 .5  gc buffer busy release 60 12 127 2113 0.0 .5  Parameter File I/O 592 0 116 195 0.0 .5  log file sequential read 1,804 0 104 58 0.0 .4 |

 Background Wait Events 后台等待事件， 数据主要来源于DBA\_HIST\_BG\_EVENT\_SUMMARY

Event 等待事件名字

Waits  该等待事件在快照时间内等待的次数

%Timeouts :  每一个等待事件有其超时的设置，例如buffer busy waits 一般为3秒， Write Complete Waits的 timeout为1秒，如果等待事件 单次等待达到timeout的时间，则会进入下一次该等待事件

Total Wait Time  该等待事件 总的消耗的时间 ，单位为秒

Avg wait(ms): 该等待事件的单次平均等待时间，单位为毫秒

Waits/Txn: 该等待事件的等待次数和事务比

## 2-5 Operating System Statistics

|  |
| --- |
| Operating System Statistics         Snaps: 70719-70723  TIME statistic values are diffed.  All others display actual values. End Value is displayed if different  -> ordered by statistic type (CPU Use, Virtual Memory, Hardware Config), Name  Statistic Value End Value  ------------------------- ---------------------- ----------------  BUSY\_TIME 2,894,855  IDLE\_TIME 5,568,240  IOWAIT\_TIME 18,973  SYS\_TIME 602,532  USER\_TIME 2,090,082  LOAD 8 13  VM\_IN\_BYTES 0  VM\_OUT\_BYTES 0  PHYSICAL\_MEMORY\_BYTES 101,221,343,232  NUM\_CPUS 24  NUM\_CPU\_CORES 12  NUM\_CPU\_SOCKETS 2  GLOBAL\_RECEIVE\_SIZE\_MAX 4,194,304  GLOBAL\_SEND\_SIZE\_MAX 2,097,152  TCP\_RECEIVE\_SIZE\_DEFAULT 87,380  TCP\_RECEIVE\_SIZE\_MAX 4,194,304  TCP\_RECEIVE\_SIZE\_MIN 4,096  TCP\_SEND\_SIZE\_DEFAULT 16,384  TCP\_SEND\_SIZE\_MAX 4,194,304  TCP\_SEND\_SIZE\_MIN 4,096  ------------------------------------------------------------- |

Operating System Statistics   操作系统统计信息

数据来源于V$OSSTAT  / DBA\_HIST\_OSSTAT，,  TIME相关的指标单位均为百分之一秒

|  |  |
| --- | --- |
| 统计项 | 描述 |
| NUM\_CPU\_SOCKETS | 物理CPU的数目 |
| NUM\_CPU\_CORES | CPU的核数 |
| NUM\_CPUS | 逻辑CPU的数目 |
| SYS\_TIME | 在内核态被消耗掉的CPU时间片，单位为百分之一秒 |
| USER\_TIME | 在用户态被消耗掉的CPU时间片，单位为百分之一秒 |
| BUSY\_TIME | Busy\_Time=SYS\_TIME+USER\_TIME 消耗的CPU时间片，单位为百分之一秒 |
| AVG\_BUSY\_TIME | AVG\_BUSY\_TIME= BUSY\_TIME/NUM\_CPUS |
| IDLE\_TIME | 空闲的CPU时间片，单位为百分之一秒 |
| 所有CPU所能提供总的时间片 | BUSY\_TIME + IDLE\_TIME = ELAPSED\_TIME \* CPU\_COUNT |
| OS\_CPU\_WAIT\_TIME | 进程等OS调度的时间，cpu queuing |
| VM\_IN\_BYTES | 换入页的字节数 |
| VM\_OUT\_BYTES | 换出页的字节数，部分版本下并不准确，例如Bug 11712010 Abstract: VIRTUAL MEMORY PAGING ON 11.2.0.2 DATABASES，仅供参考 |
| IOWAIT\_TIME | 所有CPU花费在等待I/O完成上的时间  单位为百分之一秒 |
| RSRC\_MGR\_CPU\_WAIT\_TIME | 是指当resource manager控制CPU调度时，需要控制对应进程暂时不使用CPU而进程到内部运行队列中，以保证该进程对应的consumer group(消费组)没有消耗比指定resource manager指令更多的CPU。RSRC\_MGR\_CPU\_WAIT\_TIME指等在内部运行队列上的时间，在等待时不消耗CPU |

## 2-6 Service Statistcs

|  |
| --- |
| Service Statistics Snaps: 70719-70723  -> ordered by DB Time  Physical Logical  Service Name DB Time (s) DB CPU (s) Reads (K) Reads (K)  ---------------------------- ------------ ------------ ------------ ------------  itms-contentmasterdb-prod 897,099 20,618 35,668 1,958,580  SYS$USERS 4,312 189 5,957 13,333  itmscmp 1,941 121 14,949 18,187  itscmp 331 20 114 218  itscmp\_dgmgrl 121 1 0 0  SYS$BACKGROUND 0 0 142 30,022  ITSCMP1\_PR 0 0 0 0  its-reference-prod 0 0 0 0  itscmpXDB 0 0 0 0 |

按照Service Name来分组时间模型和 物理、逻辑读取， 部分数据来源于 WRH$\_SERVICE\_NAME;

Service Name  对应的服务名  (v$services)， SYS$BACKGROUND代表后台进程， SYS$USERS一般是系统用户登录

DB TIME (s):  本服务名所消耗的DB TIME时间，单位为秒

DB CPU(s):  本服务名所消耗的DB CPU 时间，单位为秒

Physical Reads : 本服务名所消耗的物理读

Logical Reads : 本服务所消耗的逻辑读

## 2-7  Service Wait Class Stats

|  |
| --- |
| Service Wait Class Stats Snaps: 70719-70723  -> Wait Class info for services in the Service Statistics section.  -> Total Waits and Time Waited displayed for the following wait  classes: User I/O, Concurrency, Administrative, Network  -> Time Waited (Wt Time) in seconds  Service Name  ----------------------------------------------------------------  User I/O User I/O Concurcy Concurcy Admin Admin Network Network  Total Wts Wt Time Total Wts Wt Time Total Wts Wt Time Total Wts Wt Time  --------- --------- --------- --------- --------- --------- --------- ---------  itms-contentmasterdb-prod  33321670 71443 678373 113759 0 0 1.718E+08 127  SYS$USERS  173233 3656 6738 30 2 0 72674 3  itmscmp  676773 1319 1831 0 0 0 2216 0  itscmp  219577 236 1093 0 0 0 18112 0  itscmp\_dgmgrl  34 0 8 0 0 0 9 0  SYS$BACKGROUND  71940 1300 320677 56 0 0 442252 872  ------------------------------------------------------------- |

User I/O Total Wts : 对应该服务名下 用户I/O类等待的总的次数

User I/O Wt Time : 对应该服务名下 用户I/O累等待的总时间，单位为 1/100秒

Concurcy Total Wts: 对应该服务名下 Concurrency 类型等待的总次数

Concurcy Wt Time :对应该服务名下 Concurrency 类型等待的总时间， 单位为 1/100秒

Admin Total Wts: 对应该服务名下Admin 类等待的总次数

Admin Wt Time: 对应该服务名下Admin类等待的总时间，单位为 1/100秒

Network Total Wts : 对应服务名下Network类等待的总次数

Network Wt Time： 对应服务名下Network类等待的总事件， 单位为 1/100秒

## 2-8 Host CPU

|  |
| --- |
| Host CPU (CPUs:   24 Cores:   12 Sockets:    2)  ~~~~~~~~         Load Average                 Begin       End     %User   %System      %WIO     %Idle             --------- --------- --------- --------- --------- ---------                  8.41     12.84      24.7       7.1       0.2      65.8 |

“Load Average”  begin/end值代表每个CPU的大致运行队列大小。上例中快照开始到结束，平均 CPU负载增加了；与《2-5 Operating System Statistics》中的LOAD相呼应。

%User+%System=> 总的CPU使用率，在这里是31.8%

Elapsed Time \* NUM\_CPUS \* CPU utilization= 60.23 (mins)  \* 24 \* 31.8% = 459.67536 mins=Busy Time

## 2-8 Instance CPU

|  |
| --- |
| Instance CPU  ~~~~~~~~~~~~  % of total CPU for Instance: 26.7  % of busy CPU for Instance: 78.2  %DB time waiting for CPU - Resource Mgr: 0.0 |

%Total CPU,该实例所使用的CPU占总CPU的比例  % of total CPU for Instance

%Busy CPU，该实例所使用的Cpu占总的被使用CPU的比例  % of busy CPU for Instance

例如共4个逻辑CPU，其中3个被完全使用，3个中的1个完全被该实例使用，则%Total CPU= ¼ =25%，而%Busy CPU= 1/3= 33%

当CPU高时一般看%Busy CPU可以确定CPU到底是否是本实例消耗的，还是主机上其他程序

% of busy CPU for Instance= （DB CPU+ background cpu time) / (BUSY\_TIME /100)= (20,649.1  + 1,980.9)/ (2,894,855 /100)= 78.17%

% of Total CPU for Instance = ( DB CPU+ background cpu time)/( BUSY\_TIME+IDLE\_TIME/100) = (20,649.1  + 1,980.9)/ ((2,894,855+5,568,240) /100) = 26.73%

%DB time waiting for CPU (Resource Manager)= (RSRC\_MGR\_CPU\_WAIT\_TIME/100)/DB TIME

# 3 TOP SQL

TOP SQL 的数据部分来源于 dba\_hist\_sqlstat

## 3-1 SQL ordered by Elapsed Time ，按照SQL消耗的时间来排列TOP SQL

|  |
| --- |
| SQL ordered by Elapsed Time Snaps: 70719-70723  -> Resources reported for PL/SQL code includes the resources used by all SQL  statements called by the code.  -> % Total DB Time is the Elapsed Time of the SQL statement divided  into the Total Database Time multiplied by 100  -> %Total - Elapsed Time as a percentage of Total DB time  -> %CPU - CPU Time as a percentage of Elapsed Time  -> %IO - User I/O Time as a percentage of Elapsed Time  -> Captured SQL account for 53.9% of Total DB Time (s): 883,542  -> Captured PL/SQL account for 0.5% of Total DB Time (s): 883,542  Elapsed Elapsed Time  Time (s) Executions per Exec (s) %Total %CPU %IO SQL Id  ---------------- -------------- ------------- ------ ------ ------ -------------  181,411.3 38,848 4.67 20.5 .0 .1 g0yc9szpuu068 |

注意对于PL/SQL，SQL Statistics不仅会体现该PL/SQL的执行情况，还会包括该PL/SQL包含的SQL语句的情况。如上例一个TOP PL/SQL执行了448s，而这448s中绝大多数是这个PL/SQL下的一个SQL执行500次耗费的。

则该TOP PL/SQL和TOP SQL都上榜，一个执行一次耗时448s，一个执行500次耗时448s。 如此情况则Elapsed Time加起来可能超过100%的Elapsed Time，这是正常的。

对于鹤立鸡群的SQL很有必要一探究竟，跑个@?/rdbms/admin/awrsqrpt看看吧！

Elapsed Time (s): 该SQL累计运行所消耗的时间，

Executions :  该SQL在快照时间内 总计运行的次数    ；  注意， 对于在快照时间内还没有执行完的SQL 不计为1一次，所以如果看到executions=0而 又是TOP SQL，则很有可能是因为该SQL 运行较旧还没执行完，需要特别关注一下。

Elapsed Time per Exec (s)：平均每次执行该SQL耗费的时间 ， 对于OLTP类型的SELECT/INSERT/UPDATE/DELETE而言平均单次执行时间应当非常短，如0.1秒 或者更短才能满足其业务需求，如果这类轻微的OLTP操作单次也要几秒钟的话，是无法满足对外业务的需求的； 例如你在ATM上提款，并不仅仅是对你的账务库的简单UPDATE，而需要在类似风险控制的前置系统中记录你本次的流水操作记录，实际取一次钱可能要有几十乃至上百个OLTP类型的语句被执行，但它们应当都是十分快速的操作； 如果这些操作也变得很慢，则会出现大量事务阻塞，系统负载升高，DB TIME急剧上升的现象。  对于OLTP数据库而言 如果执行计划稳定，那么这些OLTP操作的性能应当是铁板钉钉的，但是一旦某个因素 发生变化，例如存储的明显变慢、内存换页的大量出现时 则上述的这些transaction操作很可能成数倍到几十倍的变慢，这将让此事务系统短期内不可用。

对于维护操作，例如加载或清除数据，大的跑批次、报表而言 Elapsed Time per Exec (s)高一些是正常的。

%Total  该SQL所消耗的时间占总的DB Time的百分比， 即 (SQL Elapsed Time / Total DB TIME)

% CPU   该SQL 所消耗的CPU 时间 占 该SQL消耗的时间里的比例， 即 (SQL CPU Time / SQL Elapsed Time) ，该指标说明了该语句是否是CPU敏感的

%IO 该SQL 所消耗的I/O 时间 占 该SQL消耗的时间里的比例， 即(SQL I/O Time/SQL Elapsed Time) ，该指标说明了该语句是否是I/O敏感的

SQL Id : 通过计算SQL 文本获得的SQL\_ID ，不同的SQL文本必然有不同的SQL\_ID， 对于10g~11g而言 只要SQL文本不变那么在数据库之间 该SQL 对应的SQL\_ID应当不不变的， 12c中修改了SQL\_ID的计算方法

Captured SQL account for   53.9% of Total DB Time (s) 对于不绑定变量的应用来说Top SQL有可能失准，所以要参考本项

## 3-2  SQL ordered by CPU Time

|  |
| --- |
| SQL ordered by CPU Time Snaps: 70719-70723  -> Resources reported for PL/SQL code includes the resources used by all SQL  statements called by the code.  -> %Total - CPU Time as a percentage of Total DB CPU  -> %CPU - CPU Time as a percentage of Elapsed Time  -> %IO - User I/O Time as a percentage of Elapsed Time  -> Captured SQL account for 34.9% of Total CPU Time (s): 20,649  -> Captured PL/SQL account for 0.5% of Total CPU Time (s): 20,649  CPU CPU per Elapsed  Time (s) Executions Exec (s) %Total Time (s) %CPU %IO SQL Id  ---------- ------------ ---------- ------ ---------- ------ ------ -------------  1,545.0 1,864,424 0.00 7.5 4,687.8 33.0 65.7 8g6a701j83c8q  Module: MZIndexer  SELECT t0.BOOLEAN\_VALUE, t0.CLASS\_CODE, t0.CREATED, t0.END\_DATE, t0.PRODUCT\_ATTR  IBUTE\_ID, t0.LAST\_MODIFIED, t0.OVERRIDE\_FLAG, t0.PRICE, t0.PRODUCT\_ATTRIBUTE\_TYP  E\_ID, t0.PRODUCT\_ID, t0.PRODUCT\_PUB\_RELEASE\_TYPE\_ID, t0.PRODUCT\_VOD\_TYPE\_ID, t0.  SAP\_PRODUCT\_ID, t0.START\_DATE, t0.STRING\_VALUE FROM mz\_product\_attribute t0 WHER |

CPU TIME :   该SQL 在快照时间内累计执行所消耗的CPU 时间片，单位为s

Executions :  该SQL在快照时间内累计执行的次数

CPU per Exec (s) ：该SQL 平均单次执行所消耗的CPU时间 ，  即  ( SQL CPU TIME / SQL Executions )

%Total : 该SQL 累计消耗的CPU时间 占  该时段总的 DB CPU的比例，  即 ( SQL CPU TIME /  Total DB CPU)

% CPU   该SQL 所消耗的CPU 时间 占 该SQL消耗的时间里的比例， 即 (SQL CPU Time / SQL Elapsed Time) ，该指标说明了该语句是否是CPU敏感的

%IO 该SQL 所消耗的I/O 时间 占 该SQL消耗的时间里的比例， 即(SQL I/O Time/SQL Elapsed Time) ，该指标说明了该语句是否是I/O敏感的

## 3-3 Buffer Gets SQL ordered by Gets

|  |
| --- |
| SQL ordered by Gets DB/Inst: ITSCMP/itscmp2 Snaps: 70719-70723  -> Resources reported for PL/SQL code includes the resources used by all SQL  statements called by the code.  -> %Total - Buffer Gets as a percentage of Total Buffer Gets  -> %CPU - CPU Time as a percentage of Elapsed Time  -> %IO - User I/O Time as a percentage of Elapsed Time  -> Total Buffer Gets: 2,021,476,421  -> Captured SQL account for 68.2% of Total  Buffer Gets Elapsed  Gets Executions per Exec %Total Time (s) %CPU %IO SQL Id  ----------- ----------- ------------ ------ ---------- ------ ------ -----------  4.61155E+08 1,864,424 247.3 22.8 4,687.8 33.0 65.7 8g6a701j83c |

注意 buffer gets 逻辑读是消耗CPU TIME的重要源泉， 但并不是说消耗CPU TIME的只有buffer gets。 大多数情况下 SQL order by CPU TIME 和 SQL order by buffers gets 2个部分的TOP SQL 及其排列顺序都是一样的，此种情况说明消耗最多buffer gets的 就是消耗最多CPU 的SQL ，如果我们希望降低系统的CPU使用率，那么只需要调优SQL 降低buffer gets 即可。

但也并不是100%的情况都是如此， CPU TIME的消耗者 还包括 函数运算、PL/SQL 控制、Latch /Mutex 的Spin等等， 所以SQL order by CPU TIME 和 SQL order by buffers gets 2个部分的TOP SQL 完全不一样也是有可能的， 需要因地制宜来探究到底是什么问题导致的High CPU，进而裁度解决之道。

Buffer Gets : 该SQL在快照时间内累计运行所消耗的buffer gets，包括了consistent read 和 current read

Executions :  该SQL在快照时间内累计执行的次数

Gets  per Exec : 该SQL平均单次的buffer gets ， 对于事务型transaction操作而言 一般该单次buffer gets小于2000

% Total  该SQL 累计运行所消耗的buffer gets占 总的db buffer gets的比率， (SQL buffer gets / DB total buffer gets)

## 3-4  Physical Reads  SQL ordered by Reads

|  |
| --- |
| SQL ordered by Reads DB/Inst: ITSCMP/itscmp2 Snaps: 70719-70723  -> %Total - Physical Reads as a percentage of Total Disk Reads  -> %CPU - CPU Time as a percentage of Elapsed Time  -> %IO - User I/O Time as a percentage of Elapsed Time  -> Total Disk Reads: 56,839,035  -> Captured SQL account for 34.0% of Total  Physical Reads Elapsed  Reads Executions per Exec %Total Time (s) %CPU %IO SQL Id  ----------- ----------- ---------- ------ ---------- ------ ------ -------------  9,006,163 1 9.0062E+06 15.8 720.9 5.9 80.9 4g36tmp70h185 |

Physical reads : 该SQL累计运行所消耗的物理读

Executions :  该SQL在快照时间内累计执行的次数

Reads per Exec : 该SQL 单次运行所消耗的物理读，  (SQL Physical reads/Executions) ， 对于OLTP transaction 类型的操作而言单次一般不超过100

%Total : 该SQL 累计消耗的物理读 占  该时段总的 物理读的比例，  即 ( SQL physical read  /  Total DB physical read )

## 3-5 Executions  SQL ordered by Executions

|  |
| --- |
| SQL ordered by Executions Snaps: 70719-70723  -> %CPU - CPU Time as a percentage of Elapsed Time  -> %IO - User I/O Time as a percentage of Elapsed Time  -> Total Executions: 48,078,147  -> Captured SQL account for 50.4% of Total  Elapsed  Executions Rows Processed Rows per Exec Time (s) %CPU %IO SQL Id  ------------ --------------- -------------- ---------- ------ ------ -----------  6,327,963 11,249,645 1.8 590.5 47.8 52.7 1avv7759j8r |

按照 执行次数来排序的话，也是性能报告对比时一个重要的参考因素，因为如果TOP SQL的执行次数有明显的增长，那么 性能问题的出现也是意料之中的事情了。 当然执行次数最多的，未必便是对性能影响最大的TOP SQL

Executions :  该SQL在快照时间内累计执行的次数

Rows Processed： 该SQL在快照时间内累计执行所处理的总行数

Rows per Exec：　SQL平均单次执行所处理的行数，  这个指标在诊断一些 数据问题造成的SQL性能问题时很有用

## 3-6 Parse Calls     SQL ordered by Parse Calls

|  |
| --- |
| SQL ordered by Parse Calls Snaps: 70719-70723  -> Total Parse Calls: 2,160,124  -> Captured SQL account for 58.3% of Total  % Total  Parse Calls Executions Parses SQL Id  ------------ ------------ --------- -------------  496,475 577,357 22.98 d07gaa3wntdff |

Parse Calls : 解析调用次数， 与上文的 Load Profile中的Parse 数一样 包括 软解析soft parse和硬解析hard parse

Executions :  该SQL在快照时间内累计执行的次数

%Total Parses : 本SQL 解析调用次数 占 该时段数据库总解析次数的比率， 为 (SQL Parse Calls / Total DB Parse Calls)

## 3-7  SQL ordered by Sharable Memory

|  |
| --- |
| SQL ordered by Sharable Memory Snaps: 70719-70723  -> Only Statements with Sharable Memory greater than 1048576 are displayed  Sharable Mem (b) Executions % Total SQL Id  ---------------- ------------ -------- -------------  8,468,359 39 0.08 au89sasqfb2yn  Module: MZContentBridge  SELECT t0.ASPECT\_RATIO, t0.CREATED, t0.FILE\_EXTENSION, t0.HEIGHT, t0.VIDEO\_FILE\_  DIMENSIONS\_ID, t0.LAST\_MODIFIED, t0.NAME, t0.WIDTH FROM MZ\_VIDEO\_FILE\_DIMENSIONS  t0 WHERE (t0.HEIGHT = :1 AND t0.WIDTH = :2 ) |

SQL ordered by Sharable Memory ,    一般该部分仅列出Sharable Mem (b)为1 MB以上的SQL 对象 (Only Statements with Sharable Memory greater than 1048576 are displayed)   数据来源是 DBA\_HIST\_SQLSTAT.SHARABLE\_MEM

Shareable Mem(b):  SQL 对象所占用的共享内存使用量

Executions :  该SQL在快照时间内累计执行的次数

%Total :  该SQL 对象锁占共享内存 占总的共享内存的比率

## 3-8   SQL ordered by Version Count

Version Count  Oracle中的执行计划可以是多版本的，即对于同一个SQL语句有多个不同版本的执行计划，这些执行计划又称作子游标， 而一个SQL语句的文本可以称作一个父游标。 一个父游标对应多个子游标，产生不同子游标的原因是 SQL在被执行时无法共享之前已经生成的子游标， 原因是多种多样的，例如 在本session中做了一个优化器参数的修改 例如optimizer\_index\_cost\_adj 从100 修改到99，则本session的优化环境optimizer env将不同于之前的子游标生成环境，这样就需要生成一个新的子游标，例如：

|  |
| --- |
| SQL> create table emp as select \* from scott.emp;  Table created.  SQL> select \* from emp where empno=1;  no rows selected  SQL> select /\*+ MACLEAN \*/ \* from emp where empno=1;  no rows selected  SQL> select SQL\_ID,version\_count from V$SQLAREA WHERE SQL\_TEXT like '%MACLEAN%' and SQL\_TEXT not like '%like%';  SQL\_ID VERSION\_COUNT  ------------- -------------  bxnnm7z1qmg26 1  SQL> select count(\*) from v$SQL where SQL\_ID='bxnnm7z1qmg26';  COUNT(\*)  ----------  1  SQL> alter session set optimizer\_index\_cost\_adj=99;  Session altered.  SQL> select /\*+ MACLEAN \*/ \* from emp where empno=1;  no rows selected  SQL> select SQL\_ID,version\_count from V$SQLAREA WHERE SQL\_TEXT like '%MACLEAN%' and SQL\_TEXT not like '%like%';  SQL\_ID VERSION\_COUNT  ------------- -------------  bxnnm7z1qmg26 2  SQL> select count(\*) from v$SQL where SQL\_ID='bxnnm7z1qmg26';  COUNT(\*)  ----------  2  SQL> select child\_number ,OPTIMIZER\_ENV\_HASH\_VALUE,PLAN\_HASH\_VALUE from v$SQL where SQL\_ID='bxnnm7z1qmg26';  CHILD\_NUMBER OPTIMIZER\_ENV\_HASH\_VALUE PLAN\_HASH\_VALUE  ------------ ------------------------ ---------------  0 3704128740 3956160932  1 3636478958 3956160932 |

可以看到上述 演示中修改optimizer\_index\_cost\_adj=99 导致CBO 优化器的优化环境发生变化， 表现为不同的OPTIMIZER\_ENV\_HASH\_VALUE，之后生成了2个子游标，但是这2个子游标的PLAN\_HASH\_VALUE同为3956160932，则说明了虽然是不同的子游标但实际子游标里包含了的执行计划是一样的；  所以请注意 任何一个优化环境的变化 (V$SQL\_SHARED\_CURSOR)以及相关衍生的BUG 都可能导致子游标无法共享，虽然子游标无法共享但这些子游标扔可能包含完全一样的执行计划，这往往是一种浪费。

注意V$SQLAREA.VERSION\_COUNT 未必等于select count(\*) FROM V$SQL WHERE SQL\_ID=”  ，即 V$SQLAREA.VERSION\_COUNT 显示的子游标数目 未必等于当前实例中还存有的子游标数目， 由于shared pool aged out算法和其他一些可能导致游标失效的原因存在，所以子游标被清理掉是很常见的事情。 V$SQLAREA.VERSION\_COUNT只是一个计数器，它告诉我们曾经生成了多少个child cursor，但不保证这些child 都还在shared pool里面。

此外可以通过v$SQL的child\_number字段来分析该问题，如果child\_number存在跳号则也说明了部分child被清理了。

子游标过多的影响， 当子游标过多(例如超过3000个时),进程需要去扫描长长的子游标列表child cursor list以找到一个合适的子游标child cursor，进而导致cursor sharing 性能问题 现大量的Cursor: Mutex S 和 library cache lock等待事件。

关于子游标的数量控制，可以参考[《11gR2游标共享新特性带来的一些问题以及\_cursor\_features\_enabled、\_cursor\_obsolete\_threshold和106001 event》](http://www.askmaclean.com/archives/11-2-obsolete-parent-cursors-_cursor_features_enabled-106001-event.html)。

Executions :  该SQL在快照时间内累计执行的次数

Hash Value :  共享SQL 的哈希值

Only Statements with Version Count greater than 20 are displayed    注意该环节仅列出version count > 20的语句

## 3-9   Cluster Wait Time SQL ordered by Cluster Wait Time

|  |
| --- |
| SQL ordered by Cluster Wait Time DB/Inst: ITSCMP/itscmp2 Snaps: 70719-70723  -> %Total - Cluster Time as a percentage of Total Cluster Wait Time  -> %Clu - Cluster Time as a percentage of Elapsed Time  -> %CPU - CPU Time as a percentage of Elapsed Time  -> %IO - User I/O Time as a percentage of Elapsed Time  -> Only SQL with Cluster Wait Time > .005 seconds is reported  -> Total Cluster Wait Time (s): 525,480  -> Captured SQL account for 57.2% of Total  Cluster Elapsed  Wait Time (s) Executions %Total Time(s) %Clu %CPU %IO SQL Id  -------------- ------------ ------ ---------- ------ ------ ------ -------------  132,639.3 38,848 25.2 181,411.3 73.1 .0 .1 g0yc9szpuu068 |

Only SQL with Cluster Wait Time > .005 seconds is reported  这个环节仅仅列出Cluster Wait Time > 0.005 s的SQL

该环节的数据主要来源 于 DBA\_HIST\_SQLSTAT.CLWAIT\_DELTA Delta value of cluster wait time

Cluster Wait Time :   该SQL语句累计执行过程中等待在集群等待上的时间，单位为秒， 你可以理解为 当一个SQL 执行过程中遇到了gc buffer busy、gc cr multi block request 之类的Cluster等待，则这些等待消耗的时间全部算在 Cluster Wait Time里。

Executions :  该SQL在快照时间内累计执行的次数

%Total:  该SQL所消耗的Cluster Wait time 占 总的Cluster Wait time的比率， 为(SQL cluster wait time / DB total cluster Wait Time)

%Clu: 该SQL所消耗的Cluster Wait time 占该SQL 总的耗时的比率，为(SQL cluster wait time / SQL elapsed Time),该指标说明了该语句是否是集群等待敏感的

% CPU   该SQL 所消耗的CPU 时间 占 该SQL消耗的时间里的比例， 即 (SQL CPU Time / SQL Elapsed Time) ，该指标说明了该语句是否是CPU敏感的

%IO 该SQL 所消耗的I/O 时间 占 该SQL消耗的时间里的比例， 即(SQL I/O Time/SQL Elapsed Time) ，该指标说明了该语句是否是I/O敏感的

# 4 Instance Activity Stats

|  |
| --- |
| Instance Activity Stats DB/Inst: ITSCMP/itscmp2 Snaps: 70719-70723  -> Ordered by statistic name  Statistic Total per Second per Trans  -------------------------------- ------------------ -------------- -------------  Batched IO (bound) vector count 450,449 124.6 1.8  Batched IO (full) vector count 5,485 1.5 0.0  Batched IO (space) vector count 1,467 0.4 0.0  Batched IO block miss count 4,119,070 1,139.7 16.7  Batched IO buffer defrag count 39,710 11.0 0.2  Batched IO double miss count 297,357 82.3 1.2  Batched IO same unit count 1,710,492 473.3 7.0  Batched IO single block count 329,521 91.2 1.3  Batched IO slow jump count 47,104 13.0 0.2  Batched IO vector block count 2,069,852 572.7 8.4  Batched IO vector read count 262,161 72.5 1.1  Block Cleanout Optim referenced 37,574 10.4 0.2  CCursor + sql area evicted 1,457 0.4 0.0  ............... |

Instance Activity Stats  的数据来自于 DBA\_HIST\_SYSSTAT，DBA\_HIST\_SYSSTAT来自于V$SYSSTAT。

这里每一个指标都代表一种数据库行为的活跃度，例如redo size 是指生成redo的量，sorts (disk) 是指磁盘排序的次数，table scans (direct read)  是指直接路径扫描表的次数。

虽然这些指标均只有Total、per Second每秒、 per Trans每事务 三个维度，但对诊断问题十分有用。

我们来举几个例子：

1、 例如当 Top Event 中存在direct path read为Top 等待事件， 则需要分清楚是对普通堆表的direct read还是由于大量LOB读造成的direct path read， 这个问题可以借助 table scans (direct read)、table scans (long tables)、physical reads direct   、physical reads direct (lob) 、physical reads direct temporary几个指标来分析， 假设 physical reads direct   >> 远大于 physical reads direct (lob)+physical reads direct temporary ， 且有较大的table scans (direct read)、table scans (long tables)  (注意这2个指标代表的是 扫描表的次数 不同于上面的phsical reads 的单位为 块数\*次数)， 则说明了是 大表扫描引起的direct path read。

2、 例如当 Top Event中存在enq Tx:index contention等待事件， 则需要分析root node splits   、branch node splits   、leaf node 90-10 splits   、leaf node splits 、failed probes on index block rec 几个指标，[具体可以见文档《Oracle索引块分裂split信息汇总》](http://www.askmaclean.com/archives/index-split.html)

3、系统出现IO类型的等待事件为TOp Five 例如 db file sequential/scattered read ，我们需要通过AWR来获得系统IO吞吐量和IOPS:

physical read bytes 主要是应用造成的物理读取(Total size in bytes of all disk reads by application activity (and not other instance activity) only.) 而physical read total bytes则包括了 rman备份恢复 和后台维护任务所涉及的物理读字节数，所以我们在研究IO负载时一般参考 physical read total bytes；以下4对指标均存在上述的关系

|  |  |  |
| --- | --- | --- |
| physical read bytes | physical read total bytes | 物理读的吞吐量/秒 |
| physical read IO requests | physical read total IO requests | 物理读的IOPS |
| physical write bytes | physical write total bytes | 物理写的吞吐量/秒 |
| physical write IO requests | physical write total IO requests | 物理写的IOPS |

总的物理吞吐量/秒=physical read total bytes+physical write total bytes

总的物理IOPS= physical read total IO requests+ physical write total IO requests

IO的主要指标 吞吐量、IOPS和延迟 均可以从AWR中获得了， IO延迟的信息可以从 User I/O的Wait Class Avg Wait time获得，也可以参考11g出现的IOStat by Function summary

Instance Activity Stats有大量的指标，但是对于这些指标的介绍 没有那一份文档有完整详尽的描述，即便在Oracle原厂内部要没有(或者是Maclean没找到)，实际是开发人员要引入某一个Activity Stats是比较容易的，并不像申请引入一个新后台进程那样麻烦，Oracle对于新版本中新后台进程的引入有严格的要求，但Activity Stats却很容易，往往一个one-off patch中就可以引入了，实际上Activity Stats在源代码层仅仅是一些计数器。’

[较为基础的statistics，大家可以参考官方文档的Statistics Descriptions描述，地址在这里。](http://docs.oracle.com/cd/E11882_01/server.112/e25513/stats002.htm)

对于深入的指标 例如  “Batched IO (space) vector count”这种由于某些新特性被引入的，一般没有很详细的材料，需要到源代码中去阅读相关模块才能总结其用途，对于这个工作一般原厂是很延迟去完成的，所以没有一个完整的列表。 如果大家有对此的疑问，请去t.askmaclean.com 发一个帖子提问。

|  |
| --- |
| Instance Activity Stats - Absolute Values Snaps: 7071  -> Statistics with absolute values (should not be diffed)  Statistic Begin Value End Value  -------------------------------- --------------- ---------------  session pga memory max 1.157882826E+12 1.154290304E+12  session cursor cache count 157,042,373 157,083,136  session uga memory 5.496429019E+14 5.496775467E+14  opened cursors current 268,916 265,694  workarea memory allocated 827,704 837,487  logons current 2,609 2,613  session uga memory max 1.749481584E+13 1.749737418E+13  session pga memory 4.150306913E+11 4.150008177E+11 |

Instance Activity Stats – Absolute Values是显示快照 起点 和终点的一些指标的绝对值

logon current 当前时间点的登录数

opened cursors current 当前打开的游标数

session cursor cache count 当前存在的session缓存游标数

|  |
| --- |
| Instance Activity Stats - Thread ActivityDB/Inst: G10R25/G10R25 Snaps: 3663-3  -> Statistics identified by '(derived)' come from sources other than SYSSTAT  Statistic Total per Hour  -------------------------------- ------------------ ---------  log switches (derived) 17 2,326.47 |

log switches (derived) 日志切换次数 ， [见 《理想的在线重做日志切换时间是多长？》](http://t.askmaclean.com/thread-3094-1-1.html)

# 5 IO 统计

## 5-1 Tablespace IO Stats  基于表空间分组的IO信息

|  |
| --- |
| Tablespace IO Stats DB/Inst: ITSCMP/itscmp2 Snaps: 70719-70723  -> ordered by IOs (Reads + Writes) desc  Tablespace  ------------------------------  Av Av Av Av Buffer Av Buf  Reads Reads/s Rd(ms) Blks/Rd Writes Writes/s Waits Wt(ms)  -------------- ------- ------- ------- ------------ -------- ---------- -------  DATA\_TS  17,349,398 4,801 2.3 1.5 141,077 39 4,083,704 5.8  INDEX\_TS  9,193,122 2,544 2.0 1.0 238,563 66 3,158,187 46.1  UNDOTBS1  1,582,659 438 0.7 1.0 2 0 12,431 69.0 |

reads : 指 该表空间上发生的物理读的次数(单位不是块，而是次数)

Av Reads/s : 指该表空间上平均每秒的物理读次数 (单位不是块，而是次数)

Av Rd(ms): 指该表空间上每次读的平均读取延迟

Av Blks/Rd: 指该表空间上平均每次读取的块数目，因为一次物理读可以读多个数据块；如果Av Blks/Rd>>1 则可能系统有较多db file scattered read 可能是诊断FULL TABLE SCAN或FAST FULL INDEX SCAN，需要关注table scans (long tables) 和index fast full scans (full)   2个指标

Writes : 该表空间上发生的物理写的次数 ;  对于那些Writes总是等于0的表空间 不妨了解下是否数据为只读，如果是可以通过read only tablespace来解决 RAC中的一些性能问题。

Av Writes/s  : 指该表空间上平均每秒的物理写次数

buffer Waits:  该表空间上发生buffer busy waits和read by other session的次数( 9i中buffer busy waits包含了read by other session)。

Av Buf Wt(ms):  该表空间上发生buffer Waits的平均等待时间，单位为ms

## 5-2 File I/O

|  |
| --- |
| File IO Stats Snaps: 70719-70723  -> ordered by Tablespace, File  Tablespace Filename  ------------------------ ----------------------------------------------------  Av Av Av Av Buffer Av Buf  Reads Reads/s Rd(ms) Blks/Rd Writes Writes/s Waits Wt(ms)  -------------- ------- ------- ------- ------------ -------- ---------- -------  AMG\_ALBUM\_IDX\_TS +DATA/itscmp/plugged/data2/amg\_album\_idx\_ts01.dbf  23,298 6 0.6 1.0 2 0 0 0.0  AMG\_ALBUM\_IDX\_TS +DATA/itscmp/plugged/data3/amg\_album\_idx\_ts02.dbf  3,003 1 0.6 1.0 2 0 0 0.0 |

Tablespace 表空间名

FileName  数据文件的路径

Reads: 该数据文件上累计发生过的物理读次数，不是块数

Av Reads/s: 该数据文件上平均每秒发生过的物理读次数，不是块数

Av Rd(ms): 该数据文件上平均每次物理读取的延迟，单位为ms

Av Blks/Rd:  该数据文件上平均每次读取涉及到的块数，OLTP环境该值接近 1

Writes : 该数据文件上累计发生过的物理写次数，不是块数

Av Writes/s: 该数据文件上平均每秒发生过的物理写次数，不是块数

buffer Waits:  该数据文件上发生buffer busy waits和read by other session的次数( 9i中buffer busy waits包含了read by other session)。

Av Buf Wt(ms):  该数据文件上发生buffer Waits的平均等待时间，单位为ms

若某个表空间上有较高的IO负载，则有必要分析一下 是否其所属的数据文件上的IO 较为均匀 还是存在倾斜， 是否需要结合存储特征来 将数据均衡分布到不同磁盘上的数据文件上，以优化 I/O

# 6 缓冲池统计 Buffer Pool Statistics

|  |
| --- |
| Buffer Pool Statistics Snaps: 70719-70723  -> Standard block size Pools D: default, K: keep, R: recycle  -> Default Pools for other block sizes: 2k, 4k, 8k, 16k, 32k  Free Writ Buffer  Number of Pool Buffer Physical Physical Buff Comp Busy  P Buffers Hit% Gets Reads Writes Wait Wait Waits  --- ---------- ---- ------------ ------------ ----------- ------ ------ --------  16k 15,720 N/A 0 0 0 0 0 0  D 2,259,159 98 2.005084E+09 42,753,650 560,460 0 1 8.51E+06 |

该环节的数据主要来源于WRH$\_BUFFER\_POOL\_STATISTICS， 而WRH$\_BUFFER\_POOL\_STATISTICS是定期汇总v$SYSSTAT中的数据

P   pool池的名字   D: 默认的缓冲池 default buffer pool  , K : Keep Pool , R: Recycle Pool ;  2k 4k 8k  16k 32k: 代表各种非标准块大小的缓冲池

Number of buffers:  实际的 缓冲块数目，   约等于  池的大小 / 池的块大小

Pool Hit % :  该缓冲池的命中率

Buffer Gets: 对该缓冲池的中块的访问次数 包括  consistent gets 和 db block gets

Physical Reads: 该缓冲池Buffer Cache引起了多少物理读， 其实是physical reads cache ，单位为 块数\*次数

Physical Writes ：该缓冲池中Buffer cache被写的物理写， 其实是physical writes from cache， 单位为 块数\*次数

Free Buffer Waits:  等待空闲缓冲的次数， 可以看做该buffer pool 发生free buffer waits 等待的次数

Write Comp Wait:   等待DBWR写入脏buffer到磁盘的次数， 可以看做该buffer pool发生write complete waits等待的次数

Buffer Busy Waits:  该缓冲池发生buffer busy wait 等待的次数

## 7-1 Checkpoint Activity  检查点与 Instance Recovery Stats    实例恢复

|  |
| --- |
| Checkpoint Activity Snaps: 70719-70723  -> Total Physical Writes: 590,563  Other Autotune Thread  MTTR Log Size Log Ckpt Settings Ckpt Ckpt  Writes Writes Writes Writes Writes Writes  ----------- ----------- ----------- ----------- ----------- -----------  0 0 0 0 12,899 0  -------------------------------------------------------------  Instance Recovery Stats Snaps: 70719-70723  -> B: Begin Snapshot, E: End Snapshot  Estd  Targt Estd Log Ckpt Log Ckpt Opt RAC  MTTR MTTR Recovery Actual Target Log Sz Timeout Interval Log Avail  (s) (s) Estd IOs RedoBlks RedoBlks RedoBlks RedoBlks RedoBlks Sz(M) Time  - ----- ----- -------- -------- -------- -------- -------- -------- ------ -----  B 0 6 12828 477505 1786971 5096034 1786971 N/A N/A 3  E 0 7 16990 586071 2314207 5096034 2314207 N/A N/A 3  ------------------------------------------------------------- |

该环节的数据来源于WRH$\_INSTANCE\_RECOVERY

MTTR  Writes  :   为了满足FAST\_START\_MTTR\_TARGET 指定的MTTR值 而做出的物理写  WRITES\_MTTR

Log Size Writes ：由于最小的redo log file而做出的物理写 WRITES\_LOGFILE\_SIZE

Log Ckpt writes: 由于 LOG\_CHECKPOINT\_INTERVAL 和 LOG\_CHECKPOINT\_TIMEOUT 驱动的增量检查点而做出的物理写 WRITES\_LOG\_CHECKPOINT\_SETTINGS

Other Settings Writes ：由于其他设置(例如FAST\_START\_IO\_TARGET）而引起的物理写，  WRITES\_OTHER\_SETTINGS

Autotune Ckpt Writes : 由于自动调优检查点而引起的物理写， WRITES\_AUTOTUNE

Thread Ckpt Writes ：由于thread checkpoint而引起的物理写，WRITES\_FULL\_THREAD\_CKPT  
B 代表 开始点， E 代表结尾

Targt MTTR (s) : 目标MTTR (mean time to recover)意为有效恢复时间，单位为秒。 TARGET\_MTTR 的计算基于 给定的参数FAST\_START\_MTTR\_TARGET，而TARGET\_MTTR作为内部使用。 实际在使用中 Target MTTR未必能和FAST\_START\_MTTR\_TARGET一样。 如果FAST\_START\_MTTR\_TARGET过小，那么TARGET\_MTTR 将是系统条件所允许的最小估算值；  如果FAST\_START\_MTTR\_TARGET过大，则TARGET\_MTTR以保守算法计算以获得完成恢复的最长估算时间。

estimated\_mttr (s):   当前基于 脏buffer和重做日志块的数量，而评估出的有效恢复时间 。 它的估算告诉用户 以当下系统的负载若发生实例crash，则需要多久时间来做crash recovery的前滚操作，之后才能打开数据库。

Recovery Estd IOs ：实际是当前buffer cache中的脏块数量，一旦实例崩溃 这些脏块要被前滚

Actual RedoBlks ：  当前实际需要恢复的redo重做块数量

Target RedoBlks ：是 Log Sz RedoBlks 、Log Ckpt  Timeout  RedoBlks、 Log Ckpt Interval  RedoBlks 三者的最小值

Log Sz RedoBlks :   代表 必须在log file switch日志切换之前完成的 checkpoint 中涉及到的redo block，也叫max log lag； 数据来源select LOGFILESZ   from X$targetrba;  select LOG\_FILE\_SIZE\_REDO\_BLKS from v$instance\_recovery;

Log Ckpt Timeout RedoBlks ： 为了满足LOG\_CHECKPOINT\_TIMEOUT  所需要处理的redo block数，lag for checkpoint timeout ； 数据来源select CT\_LAG from x$targetrba;

Log Ckpt Interval RedoBlks ：为了满足LOG\_CHECKPOINT\_INTERVAL 所需要处理的redo block数， lag for checkpoint interval； 数据来源select CI\_LAG from x$targetrba;

Opt Log Sz(M) :  基于FAST\_START\_MTTR\_TARGET 而估算出来的redo logfile 的大小，单位为MB 。 Oracle官方推荐创建的重做日志大小至少大于这个估算值

Estd RAC Avail Time  ：指评估的 RAC中节点失败后 集群从冻结到部分可用的时间， 这个指标仅在RAC中可用，单位为秒。 ESTD\_CLUSTER\_AVAILABLE\_TIME

## 7-2 Buffer Pool Advisory 缓冲池建议

|  |
| --- |
| Buffer Pool Advisory DB/Inst: ITSCMP/itscmp2 Snap: 70723  -> Only rows with estimated physical reads >0 are displayed  -> ordered by Block Size, Buffers For Estimate  Est  Phys Estimated Est  Size for Size Buffers Read Phys Reads Est Phys %DBtime  P Est (M) Factor (thousands) Factor (thousands) Read Time for Rds  --- -------- ------ ------------ ------ -------------- ------------ -------  D 1,920 .1 227 4.9 1,110,565,597 1 1.0E+09  D 3,840 .2 454 3.6 832,483,886 1 7.4E+08  D 5,760 .3 680 2.8 634,092,578 1 5.6E+08  D 7,680 .4 907 2.2 500,313,589 1 4.3E+08  D 9,600 .5 1,134 1.8 410,179,557 1 3.5E+08  D 11,520 .6 1,361 1.5 348,214,283 1 2.9E+08  D 13,440 .7 1,588 1.3 304,658,441 1 2.5E+08  D 15,360 .8 1,814 1.2 273,119,808 1 2.2E+08  D 17,280 .9 2,041 1.1 249,352,943 1 2.0E+08  D 19,200 1.0 2,268 1.0 230,687,206 1 1.8E+08  D 19,456 1.0 2,298 1.0 228,664,269 1 1.8E+08  D 21,120 1.1 2,495 0.9 215,507,858 1 1.7E+08  D 23,040 1.2 2,722 0.9 202,816,787 1 1.6E+08  D 24,960 1.3 2,948 0.8 191,974,196 1 1.5E+08  D 26,880 1.4 3,175 0.8 182,542,765 1 1.4E+08  D 28,800 1.5 3,402 0.8 174,209,199 1 1.3E+08  D 30,720 1.6 3,629 0.7 166,751,631 1 1.2E+08  D 32,640 1.7 3,856 0.7 160,002,420 1 1.2E+08  D 34,560 1.8 4,082 0.7 153,827,351 1 1.1E+08  D 36,480 1.9 4,309 0.6 148,103,338 1 1.1E+08  D 38,400 2.0 4,536 0.6 142,699,866 1 1.0E+08 |

缓冲池的颗粒大小 可以参考 SELECT \* FROM V$SGAINFO where name like(‘Granule%’);  
P 指 缓冲池的名字  可能包括 有 D default buffer pool , K  Keep Pool , R recycle Pool

Size For Est(M):  指以该尺寸的buffer pool作为评估的对象，一般是 目前current size的 10% ~ 200%，以便了解 buffer pool 增大 ~减小 对物理读的影响

Size Factor :  尺寸因子， 只 对应buffer pool 大小  对 当前设置的比例因子， 例如current\_size是 100M ， 则如果评估值是110M 那么 size Factor 就是 1.1

Buffers (thousands) ：指这个buffer pool 尺寸下的buffer 数量， 要乘以1000才是实际值

Est  Phys Read Factor ：评估的物理读因子，　例如当前尺寸的buffer pool 会引起100个物理读， 则别的尺寸的buffer pool如果引起 120个物理读， 那么 对应尺寸的Est  Phys Read Factor就是1.2

Estimated Phys Reads (thousands)：评估的物理读数目，　要乘以　1000才是实际值， 显然不同尺寸的buffer pool对应不同的评估的物理读数目

Est Phys Read Time ： 评估的物理读时间

Est %DBtime for Rds：评估的物理读占DB TIME的比率

我们 看buffer pool advisory 一般有2个目的：

在物理读较多的情况下，希望通过增加buffer pool 大小来缓解物理读等待，这是我们关注Size Factor > 1的buffer pool尺寸是否能共有效减少Est Phys Read  Factor， 如果Est Phys Read  Factor随着Size Factor 增大 而显著减少，那么说明增大buffer cache 是可以有效减少物理读的。

 在内存紧张的情况下 ，希望从buffer pool中匀出部分内存来移作他用， 但是又不希望 buffer cache变小导致 物理读增多 性能下降， 则此时 观察Est Phys Read  Factor 是否随着Size Factor 减小而 显著增大， 如果不是 则说明减少部分buffer cache 不会导致 物理读大幅增加，也就可以安心 减少 buffer cache

注意 Size Factor 和 Est Phys Read  Factor之间不是简单的 线性关系，所以需要人为介入评估得失

## 7-3 PGA Aggr Summary

|  |
| --- |
| PGA Aggr Summary Snaps: 70719-70723  -> PGA cache hit % - percentage of W/A (WorkArea) data processed only in-memory  PGA Cache Hit % W/A MB Processed Extra W/A MB Read/Written  --------------- ------------------ --------------------------  99.9 412,527 375 |

PGA Cache Hit % : 指 W/A WorkArea工作区的数据仅在内存中处理的比率， PGA缓存命中率

workarea是PGA中负责处理 排序、哈希连接和位图合并操作的区域; workarea 也叫做 SQL 作业区域

W/A  MB processes:   指 在Workarea中处理过的数据的量，单位为MB

Extra W/A MB Read/Written :  指额外从磁盘上 读写的 工作区数据， 单位为 MB

## 7-4 PGA Aggr Target Stats

|  |
| --- |
| Warning: pga\_aggregate\_target was set too low for current workload, as this  value was exceeded during this interval. Use the PGA Advisory view  to help identify a different value for pga\_aggregate\_target.  PGA Aggr Target Stats Snaps: 70719-70723  -> B: Begin Snap E: End Snap (rows dentified with B or E contain data  which is absolute i.e. not diffed over the interval)  -> Auto PGA Target - actual workarea memory target  -> W/A PGA Used - amount of memory used for all Workareas (manual + auto)  -> %PGA W/A Mem - percentage of PGA memory allocated to workareas  -> %Auto W/A Mem - percentage of workarea memory controlled by Auto Mem Mgmt  -> %Man W/A Mem - percentage of workarea memory under manual control  %PGA %Auto %Man  PGA Aggr Auto PGA PGA Mem W/A PGA W/A W/A W/A Global Mem  Target(M) Target(M) Alloc(M) Used(M) Mem Mem Mem Bound(K)  - ---------- ---------- ---------- ---------- ------ ------ ------ ----------  B 8,192 512 23,690.5 150.1 .6 100.0 .0 838,860  E 8,192 512 23,623.6 156.9 .7 100.0 .0 838,860  ------------------------------------------------------------- |

此环节的数据来源主要是 WRH$\_PGASTAT

PGA Aggr  Target(M) ：本质上就是pga\_aggregate\_target ， 当然在AMM(memory\_target)环境下 这个值可能会自动变化

Auto PGA Target(M)  : 在自动PGA 管理模式下 实际可用的工作区内存  “aggregate PGA auto target “， 因为PGA还有其他用途 ，不能全部作为workarea memory

PGA Mem Alloc(M) ：目前已分配的PGA内存， 　alloc  不等于 inuse 即分配的内存不等于在使用的内存，理论上PGA会将确实不使用的内存返回给OS(PGA memory freed back to OS) ，但是存在PGA占用大量内存而不释放的场景

在上例中 pga\_aggregate\_target 仅为8192M ，而实际processes 在 2,615~ 8000之间，如果一个进程耗费5MB的PGA 也需要 10000M的PGA ，而实际这里 PGA Mem Alloc(M)是23,690 M ，这说明 存在PGA 的过载， 需要调整pga\_aggregate\_target

W/A PGA Used(M) ：所有的工作区workarea(包括manual和 auto)使用的内存总和量， 单位为MB

%PGA W/A Mem:  分配给workarea的内存量占总的PGA的比例，  (W/A PGA Used)/PGA Mem Alloc

%Auto W/A Mem : AUTO 自动工作区管理所控制的内存(workarea\_size\_policy=AUTO) 占总的workarea内存的比例

%Man W/A Mem : MANUAL 手动工作区管理所控制的内存(workarea\_size\_policy=MANUAL)占总的workarea内存的比例

Global Mem Bound(K) : 指 在自动PGA管理模式下一个工作区所能分配的最大内存(注意 一个SQL执行过程中可能有多个工作区workarea)。 Global Mem Bound(K)这个指标在实例运行过程中将被持续性的修正，以反应数据库当时工作区的负载情况。显然在有众多活跃工作区的系统负载下相应地Global Mem Bound将会下降。 但应当保持global bound值不要小于1 MB ， 否则建议 调高pga\_aggregate\_target

## 7-5 PGA Aggr Target Histogram

|  |
| --- |
| PGA Aggr Target Histogram Snaps: 70719-70723  -> Optimal Executions are purely in-memory operations  Low High  Optimal Optimal Total Execs Optimal Execs 1-Pass Execs M-Pass Execs  ------- ------- -------------- -------------- ------------ ------------  2K 4K 262,086 262,086 0 0  64K 128K 497 497 0 0  128K 256K 862 862 0 0  256K 512K 368 368 0 0  512K 1024K 440,585 440,585 0 0  1M 2M 68,313 68,313 0 0  2M 4M 169 161 8 0  4M 8M 50 42 8 0  8M 16M 82 82 0 0  16M 32M 1 1 0 0  32M 64M 12 12 0 0  128M 256M 2 0 2 0  ------------------------------------------------------------- |

数据来源：WRH$\_SQL\_WORKAREA\_HISTOGRAM

Low Optimal： 此行所包含工作区workarea最适合内存要求的下限

High Optimal： 此行所包含工作区workarea最适合内存要求的上限

Total Execs: 在 Low Optimal~High Optimal 范围工作区内完成的总执行数

Optimal execs: optimal 执行是指完全在PGA内存中完成的执行次数

1-pass Execs :  指操作过程中仅发生1次磁盘读取的执行次数

M-pass Execs:  指操作过程中发生了1次以上的磁盘读取， 频发磁盘读取的执行次数

## 7-6 PGA Memory Advisory

|  |
| --- |
| PGA Memory Advisory Snap: 70723  -> When using Auto Memory Mgmt, minimally choose a pga\_aggregate\_target value  where Estd PGA Overalloc Count is 0  Estd Extra Estd P Estd PGA  PGA Target Size W/A MB W/A MB Read/ Cache Overallo Estd  Est (MB) Factr Processed Written to Disk Hit % Count Time  ---------- ------- ---------------- ---------------- ------ -------- -------  1,024 0.1 2,671,356,938.7 387,531,258.9 87.0 1.07E+07 7.9E+11  2,048 0.3 2,671,356,938.7 387,529,979.1 87.0 1.07E+07 7.9E+11  4,096 0.5 2,671,356,938.7 387,518,881.8 87.0 1.07E+07 7.9E+11  6,144 0.8 2,671,356,938.7 387,420,749.5 87.0 1.07E+07 7.9E+11  8,192 1.0 2,671,356,938.7 23,056,196.5 99.0 1.07E+07 6.9E+11  9,830 1.2 2,671,356,938.7 22,755,192.6 99.0 6.81E+06 6.9E+11  11,469 1.4 2,671,356,938.7 20,609,438.5 99.0 4.15E+06 6.9E+11  13,107 1.6 2,671,356,938.7 19,021,139.1 99.0 581,362 6.9E+11  14,746 1.8 2,671,356,938.7 18,601,191.0 99.0 543,531 6.9E+11  16,384 2.0 2,671,356,938.7 18,561,361.1 99.0 509,687 6.9E+11  24,576 3.0 2,671,356,938.7 18,527,422.3 99.0 232,817 6.9E+11  32,768 4.0 2,671,356,938.7 18,511,872.6 99.0 120,180 6.9E+11  49,152 6.0 2,671,356,938.7 18,500,815.3 99.0 8,021 6.9E+11  65,536 8.0 2,671,356,938.7 18,498,733.0 99.0 0 6.9E+11 |

PGA Target   Est (MB)  用以评估的 PGA\_AGGREGATE \_TARGET值

Size Factr   ， 当前用以评估的PGA\_AGGREGATE \_TARGET 和 当前实际设置的PGA\_AGGREGATE \_TARGET 之间的 比例因子  PGA Target Est / PGA\_AGGREGATE\_TARGE

W/A MB Processed ：workarea中要处理的数据量， 单位为MB

Estd Extra  W/A MB Read/ Written to Disk :   以 one-pass 、M-Pass方式处理的数据量预估值， 单位为MB

Estd P Cache Hit % :  预估的PGA缓存命中率

Estd PGA Overalloc Count: 预估的PGA过载量， 如上文所述PGA\_AGGREGATE \_TARGET仅是一个目标值，无法真正限制PGA内存的使用，当出现 PGA内存硬性需求时会产生PGA overallocate 过载(When using Auto Memory Mgmt, minimally choose a pga\_aggregate\_target value where Estd PGA Overalloc Count is 0)

## 7-7  Shared Pool Advisory

|  |
| --- |
| Shared Pool Advisory Snap: 70723  -> SP: Shared Pool Est LC: Estimated Library Cache Factr: Factor  -> Note there is often a 1:Many correlation between a single logical object  in the Library Cache, and the physical number of memory objects associated  with it. Therefore comparing the number of Lib Cache objects (e.g. in  v$librarycache), with the number of Lib Cache Memory Objects is invalid.                                         Est LC Est LC  Est LC Est LC    Shared    SP   Est LC                  Time   Time    Load   Load       Est LC      Pool  Size     Size       Est LC    Saved  Saved    Time   Time      Mem Obj   Size(M) Factr      (M)      Mem Obj      (s)  Factr     (s)  Factr     Hits (K)  -------- ----- -------- ------------ -------- ------ ------- ------ ------------       304    .8       56        3,987    7,728    1.0      61    1.4          332       352    .9      101        6,243    7,745    1.0      44    1.0          334       400   1.0      114        7,777    7,745    1.0      44    1.0          334       448   1.1      114        7,777    7,745    1.0      44    1.0          334       496   1.2      114        7,777    7,745    1.0      44    1.0          334       544   1.4      114        7,777    7,745    1.0      44    1.0          334       592   1.5      114        7,777    7,745    1.0      44    1.0          334       640   1.6      114        7,777    7,745    1.0      44    1.0          334       688   1.7      114        7,777    7,745    1.0      44    1.0          334       736   1.8      114        7,777    7,745    1.0      44    1.0          334       784   2.0      114        7,777    7,745    1.0      44    1.0          334       832   2.1      114        7,777    7,745    1.0      44    1.0          334            ------------------------------------------------------------- |

Shared  Pool  Size(M) :  用以评估的shared pool共享池大小，在AMM /ASMM环境下 shared\_pool 大小都可能浮动

SP Size Factr ：共享池大小的比例因子，　（Shared Pool Size for Estim / SHARED\_POOL\_SIZE）

Estd LC Size(M) : 评估的 library cache 大小 ，单位为MB ， 因为是shared pool中包含 library cache 当然还有其他例如row cache

Est LC Mem Obj   指评估的指定大小的共享池内的library cache memory object的数量  ESTD\_LC\_MEMORY\_OBJECTS

Est LC Time Saved(s):   指在 指定的共享池大小情况下可找到需要的library cache memory objects，从而节约的解析时间 。  这些节约的解析时间也是 花费在共享池内重复加载需要的对象(reload)，这些对象可能因为共享池没有足够的free memory而被aged out.  ESTD\_LC\_TIME\_SAVED

Est LC Time Saved Factr : Est LC Time Saved(s)的比例因子，(  Est LC Time Saved(s)/ Current LC Time Saved(s) )   ESTD\_LC\_TIME\_SAVED\_FACTOR

Est LC Load Time (s):  在指定的共享池大小情况下解析的耗时

Est LC Load Time Factr：Est LC Load Time (s)的比例因子， (Est LC Load Time (s)/ Current LC Load Time (s))         ESTD\_LC\_LOAD\_TIME\_FACTOR

Est LC  Mem Obj Hits (K) :  在指定的共享池大小情况下需要的library cache memory object正好在共享池中被找到的次数  ESTD\_LC\_MEMORY\_OBJECT\_HITS；

对于想缩小 shared\_pool\_size 共享池大小的需求，可以关注Est LC  Mem Obj Hits (K) ，如上例中共享池为352M时Est LC  Mem Obj Hits (K) 就为334且之后不动，则可以考虑缩小shared\_pool\_size到该值，但要注意每个版本/平台上对共享池的最低需求，包括RAC中gcs resource 、gcs shadow等资源均驻留在shared pool中，增大db\_cache\_size时要对应关注。

## 7-8 SGA Target Advisory

|  |
| --- |
| SGA Target Advisory Snap: 70723  SGA Target SGA Size Est DB Est Physical  Size (M) Factor Time (s) Reads  ---------- ---------- ------------ ----------------  3,752 0.1 1.697191E+09 1.4577142918E+12  7,504 0.3 1.222939E+09 832,293,601,354  11,256 0.4 1.000162E+09 538,390,923,784  15,008 0.5 895,087,191 399,888,743,900  18,760 0.6 840,062,594 327,287,716,803  22,512 0.8 806,389,685 282,881,041,331  26,264 0.9 782,971,706 251,988,446,808  30,016 1.0 765,293,424 228,664,652,276  33,768 1.1 751,135,535 210,005,616,650  37,520 1.3 739,350,016 194,387,820,900  41,272 1.4 733,533,785 187,299,216,679  45,024 1.5 732,921,550 187,299,216,679  48,776 1.6 732,691,962 187,299,216,679  52,528 1.8 732,538,908 187,299,216,679  56,280 1.9 732,538,917 187,299,216,679  60,032 2.0 732,462,391 187,299,458,716  ------------------------------------------------------------- |

该环节数据来源于WRH$\_SGA\_TARGET\_ADVICE

SGA target Size   : 用以评估的sga target大小 (sga\_target)

SGA Size Factor:  SGA Size的比例因子，  (est SGA target Size / Current SGA target Size )

Est DB Time (s): 评估对应于该指定sga target size会产生多少量的DB TIME，单位为秒

Est Physical Reads：评估对应该指定的sga target size 会产生多少的物理读

## 7-9 Streams Pool Advisory

|  |
| --- |
| Streams Pool Advisory DB/Inst: ITSCMP/itscmp2 Snap: 70723  Size for Size Est Spill Est Spill Est Unspill Est Unspill  Est (MB) Factor Count Time (s) Count Time (s)  ---------- --------- ----------- ----------- ----------- -----------  64 0.5 0 0 0 0  128 1.0 0 0 0 0  192 1.5 0 0 0 0  256 2.0 0 0 0 0  320 2.5 0 0 0 0  384 3.0 0 0 0 0  448 3.5 0 0 0 0  512 4.0 0 0 0 0  576 4.5 0 0 0 0  640 5.0 0 0 0 0  704 5.5 0 0 0 0  768 6.0 0 0 0 0  832 6.5 0 0 0 0  896 7.0 0 0 0 0  960 7.5 0 0 0 0  1,024 8.0 0 0 0 0  1,088 8.5 0 0 0 0  1,152 9.0 0 0 0 0  1,216 9.5 0 0 0 0  1,280 10.0 0 0 0 0 |

该环节只有当使用了Streams  流复制时才会有必要数据， 数据来源 WRH$\_STREAMS\_POOL\_ADVICE

Size for Est (MB) : 用以评估的 streams pool大小

Size Factor ：streams pool大小的比例因子

Est Spill Count  ：评估出的 当使用该大小的流池时 message溢出到磁盘的数量 ESTD\_SPILL\_COUNT

Est Spill Time (s)： 评估出的 当使用该大小的流池时 message溢出到磁盘的耗时，单位为秒 ESTD\_SPILL\_TIME

Est Unspill Count：评估的　当使用该大小的流池时　message unspill 即从磁盘上读取的数量 ESTD\_UNSPILL\_COUNT

Est Unspill Time (s) ： 评估的　当使用该大小的流池时 message unspill 即从磁盘上读取的耗时，单位为秒 ESTD\_UNSPILL\_TIME

## 7-10 Java Pool Advisory

java pool的相关指标与shared pool相似，不再鏖述

# 8 Wait Statistics

## 8-1 Buffer Wait Statistics

|  |
| --- |
| Buffer Wait Statistics Snaps: 70719-70723  -> ordered by wait time desc, waits desc  Class Waits Total Wait Time (s) Avg Time (ms)  ------------------ ----------- ------------------- --------------  data block 8,442,041 407,259 48  undo header 16,212 1,711 106  undo block 21,023 557 26  1st level bmb 1,038 266 256  2nd level bmb 540 185 342  bitmap block 90 25 276  segment header 197 13 66  file header block 132 6 43  bitmap index block 18 0 1  extent map 2 0 0 |

数据来源 ： WRH$\_WAITSTAT

该环节是对 缓冲池中各类型(class) 块 等待的汇总信息， wait的原因一般是 buffer busy waits 和 read by other session

class 数据块的class，  一个oracle数据块即有class 属性 还有type 属性，数据块中记录type属性(KCBH)， 而在buffer header里存有class属性(X$BH.class)

Waits: 该类型数据块的等待次数

Total Wait Time  (s) : 该类型数据块的合计等待时间 单位为秒

Avg Time (ms) : 该类型数据块 平均每次等待的耗时， 单位 ms

如果用户正使用 undo\_management=AUTO 的SMU 则一般不会因为rollback segment过少而引起undo header block类块的等待

对于INSERT 而引起的 buffer争用等待：

1、 对于手动segment 管理MSSM 考虑增加Freelists、Freelist Groups

2、 使用ASSM ，当然ASSM本身没什么参数可调

对于INSERT ON INDEX 引起的争用：

使用反向索引key

使用HASH分区和本地索引

可能的情况下 减少index的density

## 8-2 Enqueue Activity

enqueue 队列锁等待

|  |
| --- |
| Enqueue Activity Snaps: 70719-70723  -> only enqueues with waits are shown  -> Enqueue stats gathered prior to 10g should not be compared with 10g data  -> ordered by Wait Time desc, Waits desc  Enqueue Type (Request Reason)  ------------------------------------------------------------------------------  Requests Succ Gets Failed Gets Waits Wt Time (s) Av Wt Time(ms)  ------------ ------------ ----------- ----------- ------------ --------------  TX-Transaction (index contention)  201,270 201,326 0 193,948 97,517 502.80  TM-DML  702,731 702,681 4 1,081 46,671 43,174.08  SQ-Sequence Cache  28,643 28,632 0 17,418 35,606 2,044.19  HW-Segment High Water Mark  9,210 8,845 376 1,216 12,505 10,283.85  TX-Transaction (row lock contention)  9,288 9,280 0 9,232 10,486 1,135.80  CF-Controlfile Transaction  15,851 14,094 1,756 2,798 4,565 1,631.64  TX-Transaction (allocate ITL entry)  471 369 102 360 169 469.28 |

Enqueue Type (Request Reason) enqueue 队列的类型，大家在研究 enqueue 问题前 至少搞清楚enqueue type 和enqueue mode ， enqueue type是队列锁所要保护的资源 如 TM 表锁  CF 控制文件锁， enqueue mode 是持有队列锁的模式 (SS、SX 、S、SSX、X)

Requests : 申请对应的enqueue type资源或者队列转换(enqueue conversion   例如 S 转 SSX ) 的次数

Succ Gets ：对应的enqueue被成功 申请或转换的次数

Failed Gets ：对应的enqueue的申请　或者转换失败的次数

Waits　：由对应的enqueue的申请或者转换而造成等待的次数

Wt Time (s) ： 由对应的enqueue的申请或者转换而造成等待的等待时间

Av Wt Time(ms) ：由对应的enqueue的申请或者转换而造成等待的平均等待时间  ， Wt Time (s) / Waits ,单位为ms

主要的enqueue 等待事件：

[enq: TX – row lock/index contention、allocate ITL等待事件](http://www.askmaclean.com/archives/enqueue-tx-row-lock-index-itl-wait-event.html)

[enq: TM – contention等待事件](http://www.askmaclean.com/archives/enq-tm-contention.html)

[Oracle队列锁enq:TS,Temporary Segment (also TableSpace)](http://www.askmaclean.com/archives/enq-ts-temporary-segment.html)

## 9-1 Undo Segment Summary

|  |
| --- |
| Undo Segment Summary Snaps: 70719-70723  -> 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 Num Undo Number of Max Qry Max Tx Min/Max STO/ uS/uR/uU/  TS# Blocks (K) Transactions Len (s) Concurcy TR (mins) OOS eS/eR/eU  ---- ---------- --------------- -------- -------- --------- ----- --------------  4 85.0 200,127 55,448 317 1040.2/10 0/0 0/0/0/0/0/0  -------------------------------------------------------------  Undo Segment Stats Snaps: 70719-70723  -> Most recent 35 Undostat rows, ordered by Time desc  Num Undo Number of Max Qry Max Tx Tun Ret STO/ uS/uR/uU/  End Time Blocks Transactions Len (s) Concy (mins) OOS eS/eR/eU  ------------ ----------- ------------ ------- ------- ------- ----- ------------  29-Aug 05:52 11,700 35,098 55,448 234 1,070 0/0 0/0/0/0/0/0  29-Aug 05:42 12,203 24,677 54,844 284 1,065 0/0 0/0/0/0/0/0  29-Aug 05:32 14,132 37,826 54,241 237 1,060 0/0 0/0/0/0/0/0  29-Aug 05:22 14,379 32,315 53,637 317 1,050 0/0 0/0/0/0/0/0  29-Aug 05:12 15,693 34,157 53,033 299 1,045 0/0 0/0/0/0/0/0  29-Aug 05:02 16,878 36,054 52,428 250 1,040 0/0 0/0/0/0/0/0 |

数据来源：  WRH$\_UNDOSTAT   ， undo相关的使用信息每10分钟刷新到v$undostat中

Undo Extent有三种状态  active 、unexpired 、expired

active => extent中 包括了活动的事务 ，active的undo extent 一般不允许被其他事务重用覆盖

unexpired => extent中没有活动的事务，但相关undo 记录从inactive到目前还未经过undo retention(注意 auto undo retention的问题 因为这个特性 可能在观察dba\_undo\_extents时看到大部分block都是unexpired，这是正常的)  指定的时间，所以为unexpired。 对于没有guarantee retention的undo tablespace而言，unexpired extent可能被 steal 为其他事物重用

expired => extent中没有活动事务，且超过了undo retention的时间

Undo TS# 在使用的这个undo 表空间的表空间号， 一个实例 同一时间只能用1个undo tablespace ， RAC不同节点可以用不同的undo tablespace

Num Undo Blocks (K)  指被消费的 undo 数据块的数量, (K)代表要乘以1000才是实际值；  可以用该指标来评估系统对undo block的消费量， 以便基于实际负载情况来评估UNDO表空间的大小

Number of Transactions  指该段时间内该undo表空间上执行过的事务transaction总量

Max Qry Len (s)  该时段内  持续最久的查询 时间， 单位为秒

Max Tx Concy  该时段内 最大的事务并发量

Min/Max TR (mins)   最小和最大的tuned  undo retention ，单位为分钟； [tuned undo retention 是自动undo调优特性，见undo自动调优介绍。](http://www.askmaclean.com/archives/undo%E8%87%AA%E5%8A%A8%E8%B0%83%E4%BC%98%E4%BB%8B%E7%BB%8D.html)

STO/ OOS     STO 指 ORA-01555 Snapshot Too Old错误出现的次数；   OOS – 指Out of Space count 错误出现的次数

uS – unexpired Stolen  尝试从未过期的undo extent中偷取undo space的次数

uR – unexpired Released  从未过期的undo extent中释放的块数目

uU – unexpired reUsed   未过期的undo extent中的block被其他事务重用的 块数目

eS – expired   Stolen    尝试从过期的undo extent中偷取undo space的次数

eR – expired   Released   从过期的undo extent中释放的块数目

eU – expired   reUsed   过期的undo extent中的block被其他事务重用的 块数目

|  |  |  |
| --- | --- | --- |
| UNXPSTEALCNT | NUMBER | Number of attempts to obtain undo space by stealing unexpired extents from other transactions |
| UNXPBLKRELCNT | NUMBER | Number of unexpired blocks removed from certain undo segments so they can be used by other transactions |
| UNXPBLKREUCNT | NUMBER | Number of unexpired undo blocks reused by transactions |
| EXPSTEALCNT | NUMBER | Number of attempts to steal expired undo blocks from other undo segments |
| EXPBLKRELCNT | NUMBER | Number of expired undo blocks stolen from other undo segments |
| EXPBLKREUCNT | NUMBER | Number of expired undo blocks reused within the same undo segments |
| SSOLDERRCNT | NUMBER | Identifies the number of times the error ORA-01555 occurred. You can use this statistic to decide whether or not the UNDO\_RETENTION initialization parameter is set properly given the size of the undo tablespace. Increasing the value of UNDO\_RETENTION can reduce the occurrence of this error. |

## 10-1 Latch Activity

|  |
| --- |
| Latch Activity Snaps: 70719-70723  -> "Get Requests", "Pct Get Miss" and "Avg Slps/Miss" are statistics for  willing-to-wait latch get requests  -> "NoWait Requests", "Pct NoWait Miss" are for no-wait latch get requests  -> "Pct Misses" for both should be very close to 0.0  Pct Avg Wait Pct  Get Get Slps Time NoWait NoWait  Latch Name Requests Miss /Miss (s) Requests Miss  ------------------------ -------------- ------ ------ ------ ------------ ------  AQ deq hash table latch 4 0.0 0 0 N/A  ASM Keyed state latch 9,048 0.1 0.2 0 0 N/A  ASM allocation 15,017 0.2 0.8 1 0 N/A  ASM db client latch 72,745 0.0 0 0 N/A  ASM map headers 5,860 0.6 0.6 1 0 N/A  ASM map load waiting lis 1,462 0.0 0 0 N/A  ASM map operation freeli 63,539 0.1 0.4 1 0 N/A  ASM map operation hash t 76,484,447 0.1 1.0 66 0 N/A |

latch name Latch闩的名字

Get Requests      latch被以willing-to-wait模式申请并获得的次数

Pct Get Miss   miss是指latch被以willing-to-wait 模式申请但是申请者必须等待的次数，  Pct Get Miss = Miss/Get Requests  ; miss可以从后面的Latch Sleep Breakdown 获得

Avg Slps /Miss    Sleep 是指latch被以willing-to-wait模式申请最终导致session需要sleep以等待该latch的次数  ；   Avg Slps /Miss = Sleeps/ Misses ; Sleeps可以从后面的Latch Sleep Breakdown 获得

Wait Time (s)  指花费在等待latch上的时间，单位为秒

NoWait Requests  指latch被以no-wait模式来申请的次数

Pct NoWait Miss   以no-wait模式来申请latch但直接失败的次数

对于高并发的latch例如cache buffers chains，其Pct Misses应当十分接近于0

一般的调优原则：

如果latch : cache buffers chains是 Top 5 事件，则需要考虑优化SQL减少 全表扫描 并减少Top buffer gets SQL语句的逻辑读

如果latch : redo copy 、redo allocation 等待较多，则可以考虑增大LOG\_BUFFER

如果latch:library cache 发生较多，则考虑增大shared\_pool\_size

## 10-2 Latch Sleep Breakdown

|  |
| --- |
| Latch Sleep Breakdown DB/Inst: ITSCMP/itscmp2 Snaps: 70719-70723  -> ordered by misses desc  Get Spin  Latch Name Requests Misses Sleeps Gets  -------------------------- --------------- ------------ ----------- -----------  cache buffers chains 3,365,097,866 12,831,875 130,058 12,683,450  row cache objects 69,050,058 349,839 1,320 348,649  session idle bit 389,437,460 268,285 2,768 265,752  enqueue hash chains 8,698,453 239,880 22,476 219,950  ges resource hash list 8,388,730 158,894 70,728 91,104  gc element 100,383,385 135,759 6,285 129,742  gcs remastering latch 12,213,169 72,373 1 72,371  enqueues 4,662,545 46,374 259 46,155  ASM map operation hash tab 76,484,447 46,231 45,210 1,952  Lsod array latch 72,598 24,224 24,577 1,519 |

latch name Latch闩的名字

Get Requests      latch被以willing-to-wait模式申请并获得的次数

misses 是指latch被以willing-to-wait 模式申请但是申请者必须等待的次数

9i以后miss之后一般有2种情况 spin gets了 或者sleep一睡不醒直到 被post，[具体见全面解析9i以后Oracle Latch闩锁原理；](http://www.askmaclean.com/archives/oracle-latch.html)

[8i以前的latch算法可以参考：Oracle Latch:一段描绘Latch运作的伪代码](http://www.askmaclean.com/archives/oracle-latch%E4%B8%80%E6%AE%B5%E6%8F%8F%E7%BB%98latch%E8%BF%90%E4%BD%9C%E7%9A%84%E4%BC%AA%E4%BB%A3%E7%A0%81.html)

所以一般来说9i以后的 misses= Sleeps+ Spin Gets ，虽然不是绝对如此

Sleeps 是指latch被以willing-to-wait模式申请最终导致session需要sleep以等待该latch的次数

Spin Gets  以willing-to-wait模式去申请latch，在miss之后以spin方式获得了latch的次数

## 10-3 Latch Miss Sources

|  |
| --- |
| Latch Miss Sources Snaps: 70719-70723  -> only latches with sleeps are shown  -> ordered by name, sleeps desc  NoWait Waiter  Latch Name Where Misses Sleeps Sleeps  ------------------------ -------------------------- ------- ---------- --------  ASM Keyed state latch kfksolGet 0 1 1  ASM allocation kfgpnSetDisks2 0 17 0  ASM allocation kfgpnClearDisks 0 5 0  ASM allocation kfgscCreate 0 4 0  ASM allocation kfgrpGetByName 0 1 26  ASM map headers kffmUnidentify\_3 0 7 8  ASM map headers kffmAllocate 0 6 0  ASM map headers kffmIdentify 0 6 11  ASM map headers kffmFree 0 1 0  ASM map operation freeli kffmTranslate2 0 15 8  ASM map operation hash t kffmUnidentify 0 44,677 36,784  ASM map operation hash t kffmTranslate 0 220 3,517 |

数据来源为DBA\_HIST\_LATCH\_MISSES\_SUMMARY

latch name Latch闩的名字  
where  : 指哪些代码路径内核函数持有过这些该latch ，而不是哪些代码路径要申请这些latch；  例如kcbgtcr函数的作用是Get a block for Consistent read，其持有latch :cache buffers chain是很正常的事情

NoWait Misses: 以no-wait模式来申请latch但直接失败的次数

Sleeps:  指latch被以willing-to-wait模式申请最终导致session需要sleep以等待该latch的次数  time of sleeps resulted in making the latch request

Waiter Sleeps：等待者休眠的次数  times of sleeps that waiters did for each where;   Sleep 是阻塞者等待的次数 ， Waiter Sleeps是被阻塞者等待的次数

## 10-4 Mutex Sleep Summary

|  |
| --- |
| Mutex Sleep Summary Snaps: 70719-70723  -> ordered by number of sleeps desc  Wait  Mutex Type Location Sleeps Time (ms)  --------------------- -------------------------------- ------------ ------------  Cursor Pin kksfbc [KKSCHLFSP2] 4,364 14,520  Cursor Pin kkslce [KKSCHLPIN2] 2,396 2,498  Library Cache kglpndl1 95 903 475  Library Cache kglpin1 4 800 458  Library Cache kglpnal2 91 799 259  Library Cache kglget1 1 553 1,697  Library Cache kglpnal1 90 489 88  Library Cache kgllkdl1 85 481 1,528  Cursor Pin kksLockDelete [KKSCHLPIN6] 410 666  Cursor Stat kkocsStoreBindAwareStats [KKSSTA 346 497  Library Cache kglhdgn2 106 167 348  Library Cache kglhdgh1 64 26 84  Library Cache kgldtin1 42 19 55  Cursor Pin kksfbc [KKSCHLPIN1] 13 34  Library Cache kglhdgn1 62 11 13  Library Cache kgllkal1 80 9 12  Library Cache kgllkc1 57 6 0  Cursor Pin kksSetBindType [KKSCHLPIN3] 5 5  Library Cache kglGetHandleReference 124 4 20  Library Cache kglUpgradeLock 119 4 0  Library Cache kglget2 2 3 0  Library Cache kglati1 45 1 0  Library Cache kglini1 32 1 0  Library Cache kglobld1 75 1 0  Library Cache kglobpn1 71 1 0 |

Mutex是10.2.0.2以后引入的新的内存锁机制，具体对Mutex的描述见 《深入理解Oracle中的Mutex》：<http://www.askmaclean.com/archives/understanding-oracle-mutex.html>

Mutex Type

Mutex的类型其实就是 mutex对应的客户的名字，  在版本10.2中基本只有KKS使用Mutex，所以仅有3种:

Cursor Stat (kgx\_kks1)

Cursor Parent (kgx\_kks2)

Cursor Pin (kgx\_kks3)

11g中增加了Library Cache

Location  发起对该Mutex申请的代码路径code location，而不是还持有该Mutex的代码路径或曰内核函数

## 10.2中最常见的下面的几个函数

kkspsc0  -负责解析游标 – 检测我们正在解析的游标是否有对象的parent cursor heap 0存在

kksfbc           –  负责找到合适的子游标 或者创建一个新的子游标

kksFindCursorstat

Sleeps:

Mutex的Get和Sleep

当一个Mutex被申请时， 一般称为一个get request。 若初始的申请未能得到授权， 则该进程会因为此次申请而进入到255次SPIN中(\_mutex\_spin\_count Mutex spin count)，每次SPIN循环迭代过程中该进程都会去看看Mutex被释放了吗。

若该Mutex在SPIN之后仍未被释放，则该进程针对申请的mutex进入对应的mutex wait等待事件中。 实际进程的等待事件和等待方式由mutex的类型锁决定，例如 Cursor pin、Cursor Parent。  举例来说，这种等待可能是阻塞等待，也可以是sleep。

但是请注意在V$MUTEX\_SLEEP\_\*视图上的sleep列意味着等待的次数。相关代码函数在开始进入等待时自加这个sleep字段。

等待计时从进程进入等待前开始计算等待时间， 当一个进程结束其等待，则等待的时间加入都总和total中。  该进程再次尝试申请之前的Mutex，若该Mutex仍不可用，则它再次进入spin/wait的循环。

V$MUTEX\_SLEEP\_HISTORY视图的GETS列仅在成功申请到一个Mutex时才增加。

Wait  Time (ms) 类似于latch，spin time 不算做mutex的消耗时间，它只包含等待消耗的时间。

# 11  segment statistics 段级统计

## 11-1 Segments by Logical Reads

|  |
| --- |
| Segments by Logical Reads DB/Inst: MAC/MAC2 Snaps: 70719-70723  -> Total Logical Reads: 2,021,476,421  -> Captured Segments account for 83.7% of Total  Tablespace Subobject Obj. Logical  Owner Name Object Name Name Type Reads %Total  ---------- ---------- -------------------- ---------- ----- ------------ -------  CONTENT\_OW INDEX\_TS MZ\_PRODUCT\_ATTRIBUTE INDEX 372,849,920 18.44  CONTENT\_OW INDEX\_TS MZ\_PRODUCT\_\_LS\_PK INDEX 329,829,632 16.32  CONTENT\_OW DATA\_TS MZ\_PRODUCT\_ATTRIBUTE TABLE 218,419,008 10.80  CONTENT\_OW PLAYLIST\_A MZ\_PLAYLIST\_ARTIST TABLE 182,426,240 9.02  CONTENT\_OW DATA\_TS MZ\_PRODUCT TABLE 108,597,376 5.37 |

owner : 数据段的所有者

Tablespace Name: 数据段所在表空间名

Object Name : 对象名

Subobject Name：子对象名，例如一个分区表的某个分区

obj Type:  对象类型 一般为TABLE /INDEX  或者分区或子分区

Logical Reads ：该数据段上发生过的逻辑读 ， 单位为 块数\*次数

%Total : 占总的逻辑读的百分比 ，   (当前对象上发生过的逻辑读/ Total DB 逻辑读)

## 11-2 Segments by Physical Reads

|  |
| --- |
| Segments by Physical Reads DB/Inst: MAC/MAC2 Snaps: 70719-70723  -> Total Physical Reads: 56,839,035  -> Captured Segments account for 51.9% of Total  Tablespace Subobject Obj. Physical  Owner Name Object Name Name Type Reads %Total  ---------- ---------- -------------------- ---------- ----- ------------ -------  CONTENT\_OW SONG\_TS MZ\_SONG TABLE 7,311,928 12.86  CONTENT\_OW DATA\_TS MZ\_CS\_WORK\_PENDING\_R TABLE 4,896,554 8.61  CONTENT\_OW DATA\_TS MZ\_CONTENT\_PROVIDER\_ TABLE 3,099,387 5.45  CONTENT\_OW DATA\_TS MZ\_PRODUCT\_ATTRIBUTE TABLE 1,529,971 2.69  CONTENT\_OW DATA\_TS MZ\_PUBLICATION TABLE 1,391,735 2.45 |

Physical Reads: 该数据段上发生过的物理读 ， 单位为 块数\*次数

%Total : 占总的物理读的百分比 ，   (当前对象上发生过的逻辑读/ Total DB 逻辑读)

## 11-3  Segments by Physical Read Requests

|  |
| --- |
| Segments by Physical Read Requests DB/Inst: MAC/MAC2 Snaps: 70719-70723  -> Total Physical Read Requests: 33,936,360  -> Captured Segments account for 45.5% of Total  Tablespace Subobject Obj. Phys Read  Owner Name Object Name Name Type Requests %Total  ---------- ---------- -------------------- ---------- ----- ------------ -------  CONTENT\_OW DATA\_TS MZ\_CONTENT\_PROVIDER\_ TABLE 3,099,346 9.13  CONTENT\_OW DATA\_TS MZ\_PRODUCT\_ATTRIBUTE TABLE 1,529,950 4.51  CONTENT\_OW DATA\_TS MZ\_PRODUCT TABLE 1,306,756 3.85  CONTENT\_OW DATA\_TS MZ\_AUDIO\_FILE TABLE 910,537 2.68  CONTENT\_OW INDEX\_TS MZ\_PRODUCT\_ATTRIBUTE INDEX 820,459 2.42 |

Phys Read Requests ： 物理读的申请次数

%Total  ：　(该段上发生的物理读的申请次数/ physical read IO requests)

## 11-4  Segments by UnOptimized Reads

|  |
| --- |
| Segments by UnOptimized Reads DB/Inst: MAC/MAC2 Snaps: 70719-70723  -> Total UnOptimized Read Requests: 811,466  -> Captured Segments account for 58.5% of Total  Tablespace Subobject Obj. UnOptimized  Owner Name Object Name Name Type Reads %Total  ---------- ---------- -------------------- ---------- ----- ------------ -------  CONTENT\_OW DATA\_TS MZ\_CONTENT\_PROVIDER\_ TABLE 103,580 12.76  CONTENT\_OW SONG\_TS MZ\_SONG TABLE 56,946 7.02  CONTENT\_OW DATA\_TS MZ\_IMAGE TABLE 47,017 5.79  CONTENT\_OW DATA\_TS MZ\_PRODUCT\_ATTRIBUTE TABLE 40,950 5.05  CONTENT\_OW DATA\_TS MZ\_PRODUCT TABLE 30,406 3.75 |

UnOptimized Reads UnOptimized Read Reqs = Physical Read Reqts – Optimized Read Reqs

Optimized Read Requests是指 哪些满足Exadata Smart Flash Cache ( or the Smart Flash Cache in OracleExadata V2 (Note that despite same name, concept and use of  
‘Smart Flash Cache’ in Exadata V2 is different from ‘Smart Flash Cache’ in Database Smart Flash Cache)).的物理读 次数 。  满足从smart flash cache走的读取申请呗认为是optimized ，因为这些读取要比普通从磁盘走快得多。

此外通过smart scan 读取storage index的情况也被认为是’optimized read requests’ ，源于可以避免读取不相关的数据。

当用户不在使用Exadata时，则UnOptimized Read Reqs总是等于 Physical Read Reqts

%Total : (该段上发生的物理读的UnOptimized Read Reqs / ( physical read IO requests – physical read requests optimized   ))

## 11-5 Segments by Optimized Reads

|  |
| --- |
| Segments by Optimized Reads DB/Inst: MAC/MAC2 Snaps: 70719-70723  -> Total Optimized Read Requests: 33,124,894  -> Captured Segments account for 45.2% of Total  Tablespace Subobject Obj. Optimized  Owner Name Object Name Name Type Reads %Total  ---------- ---------- -------------------- ---------- ----- ------------ -------  CONTENT\_OW DATA\_TS MZ\_CONTENT\_PROVIDER\_ TABLE 2,995,766 9.04  CONTENT\_OW DATA\_TS MZ\_PRODUCT\_ATTRIBUTE TABLE 1,489,000 4.50  CONTENT\_OW DATA\_TS MZ\_PRODUCT TABLE 1,276,350 3.85  CONTENT\_OW DATA\_TS MZ\_AUDIO\_FILE TABLE 890,775 2.69  CONTENT\_OW INDEX\_TS MZ\_AM\_REQUEST\_IX3 INDEX 816,067 2.46 |

关于optimizerd read 上面已经解释过了，这里的单位是 request 次数

%Total :  (该段上发生的物理读的 Optimized Read Reqs/ physical read requests optimized )

## 11-6 Segments by Direct Physical Reads

|  |
| --- |
| Segments by Direct Physical Reads DB/Inst: MAC/MAC2 Snaps: 70719-70723  -> Total Direct Physical Reads: 14,118,552  -> Captured Segments account for 94.2% of Total  Tablespace Subobject Obj. Direct  Owner Name Object Name Name Type Reads %Total  ---------- ---------- -------------------- ---------- ----- ------------ -------  CONTENT\_OW SONG\_TS MZ\_SONG TABLE 7,084,416 50.18  CONTENT\_OW DATA\_TS MZ\_CS\_WORK\_PENDING\_R TABLE 4,839,984 34.28  CONTENT\_OW DATA\_TS MZ\_PUBLICATION TABLE 1,361,133 9.64  CONTENT\_OW DATA\_TS SYS\_LOB0000203660C00 LOB 5,904 .04  CONTENT\_OW DATA\_TS SYS\_LOB0000203733C00 LOB 1,656 .01 |

Direct reads 直接路径物理读，单位为 块数\*次数  
%Total  (该段上发生的direct path reads /Total  physical reads direct )

## 11-7  Segments by Physical Writes

|  |
| --- |
| Segments by Physical Writes DB/Inst: MAC/MAC2 Snaps: 70719-70723  -> Total Physical Writes: 590,563  -> Captured Segments account for 38.3% of Total  Tablespace Subobject Obj. Physical  Owner Name Object Name Name Type Writes %Total  ---------- ---------- -------------------- ---------- ----- ------------ -------  CONTENT\_OW DATA\_TS MZ\_CS\_WORK\_PENDING\_R TABLE 23,595 4.00  CONTENT\_OW DATA\_TS MZ\_PODCAST TABLE 19,834 3.36  CONTENT\_OW INDEX\_TS MZ\_IMAGE\_IX2 INDEX 16,345 2.77  SYS SYSAUX WRH$\_ACTIVE\_SESSION\_ 1367\_70520 TABLE 14,173 2.40  CONTENT\_OW INDEX\_TS MZ\_AM\_REQUEST\_IX3 INDEX 9,645 1.63  Physical Writes ，物理写 单位为 块数\*次数  Total % (该段上发生的物理写 /Total physical writes ) |

## 11-9 Segments by Physical Write Requests

|  |
| --- |
| Segments by Physical Write Requests DB/Inst: MAC/MAC2 Snaps: 70719-70723  -> Total Physical Write Requestss: 436,789  -> Captured Segments account for 43.1% of Total  Tablespace Subobject Obj. Phys Write  Owner Name Object Name Name Type Requests %Total  ---------- ---------- -------------------- ---------- ----- ------------ -------  CONTENT\_OW DATA\_TS MZ\_CS\_WORK\_PENDING\_R TABLE 22,581 5.17  CONTENT\_OW DATA\_TS MZ\_PODCAST TABLE 19,797 4.53  CONTENT\_OW INDEX\_TS MZ\_IMAGE\_IX2 INDEX 14,529 3.33  CONTENT\_OW INDEX\_TS MZ\_AM\_REQUEST\_IX3 INDEX 9,434 2.16  CONTENT\_OW DATA\_TS MZ\_AM\_REQUEST TABLE 8,618 1.97 |

Phys Write Requests 物理写的请求次数 ，单位为次数

%Total (该段上发生的物理写请求次数 /physical write IO requests )

## 11-10 Segments by Direct Physical Writes

|  |
| --- |
| Segments by Direct Physical Writes DB/Inst: MAC/MAC2 Snaps: 70719-70723  -> Total Direct Physical Writes: 29,660  -> Captured Segments account for 18.3% of Total  Tablespace Subobject Obj. Direct  Owner Name Object Name Name Type Writes %Total  ---------- ---------- -------------------- ---------- ----- ------------ -------  SYS SYSAUX WRH$\_ACTIVE\_SESSION\_ 1367\_70520 TABLE 4,601 15.51  CONTENT\_OW DATA\_TS SYS\_LOB0000203733C00 LOB 620 2.09  CONTENT\_OW DATA\_TS SYS\_LOB0000203660C00 LOB 134 .45  CONTENT\_OW DATA\_TS SYS\_LOB0000203779C00 LOB 46 .16  CONTENT\_OW DATA\_TS SYS\_LOB0000203796C00 LOB 41 .14 |

Direct Writes 直接路径写， 单位额为块数\*次数  
%Total 为(该段上发生的直接路径写 /physical writes direct )

## 11-11   Segments by Table Scans

|  |
| --- |
| Segments by Table Scans DB/Inst: MAC/MAC2 Snaps: 70719-70723  -> Total Table Scans: 10,713  -> Captured Segments account for 1.0% of Total  Tablespace Subobject Obj. Table  Owner Name Object Name Name Type Scans %Total  ---------- ---------- -------------------- ---------- ----- ------------ -------  CONTENT\_OW DATA\_TS MZ\_PUBLICATION TABLE 92 .86  CONTENT\_OW DATA\_TS MZ\_CS\_WORK\_PENDING\_R TABLE 14 .13  CONTENT\_OW SONG\_TS MZ\_SONG TABLE 3 .03  CONTENT\_OW DATA\_TS MZ\_AM\_REQUEST TABLE 1 .01 |

Table Scans 来源为dba\_hist\_seg\_stat.table\_scans\_delta 不过这个指标并不十分精确

## 11-12 Segments by DB Blocks Changes

|  |
| --- |
| Segments by DB Blocks Changes DB/Inst: MAC/MAC2 Snaps: 70719-70723  -> % of Capture shows % of DB Block Changes for each top segment compared  -> with total DB Block Changes for all segments captured by the Snapshot  Tablespace Subobject Obj. DB Block % of  Owner Name Object Name Name Type Changes Capture  ---------- ---------- -------------------- ---------- ----- ------------ -------  CONTENT\_OW INDEX\_TS MZ\_AM\_REQUEST\_IX8 INDEX 347,856 10.21  CONTENT\_OW INDEX\_TS MZ\_AM\_REQUEST\_IX3A INDEX 269,504 7.91  CONTENT\_OW INDEX\_TS MZ\_AM\_REQUEST\_PK INDEX 251,904 7.39  CONTENT\_OW DATA\_TS MZ\_AM\_REQUEST TABLE 201,056 5.90  CONTENT\_OW INDEX\_TS MZ\_PRODUCT\_ATTRIBUTE INDEX 199,888 5.86 |

 DB Block Changes ，单位为块数\*次数

%Total : (该段上发生block changes  /  db block changes )

## 11-13  Segments by Row Lock Waits

|  |
| --- |
| Segments by Row Lock Waits DB/Inst: MAC/MAC2 Snaps: 70719-70723  -> % of Capture shows % of row lock waits for each top segment compared  -> with total row lock waits for all segments captured by the Snapshot  Row  Tablespace Subobject Obj. Lock % of  Owner Name Object Name Name Type Waits Capture  ---------- ---------- -------------------- ---------- ----- ------------ -------  CONTENT\_OW LOB\_8K\_TS MZ\_ASSET\_WORK\_EVENT\_ INDEX 72,005 43.86  CONTENT\_OW LOB\_8K\_TS MZ\_CS\_WORK\_NOTE\_RE\_I \_2013\_1\_36 INDEX 13,795 8.40  CONTENT\_OW LOB\_8K\_TS MZ\_CS\_WORK\_INFO\_PART \_2013\_5\_35 INDEX 12,383 7.54  CONTENT\_OW INDEX\_TS MZ\_AM\_REQUEST\_IX3A INDEX 8,937 5.44  CONTENT\_OW DATA\_TS MZ\_AM\_REQUEST TABLE 8,531 5.20 |

Row  Lock Waits 是指行锁的等待次数   数据来源于 dba\_hist\_seg\_stat.ROW\_LOCK\_WAITS\_DELTA

## 11-14   Segments by ITL WAITS

|  |
| --- |
| Segments by ITL Waits DB/Inst: MAC/MAC2 Snaps: 70719-70723  -> % of Capture shows % of ITL waits for each top segment compared  -> with total ITL waits for all segments captured by the Snapshot  Tablespace Subobject Obj. ITL % of  Owner Name Object Name Name Type Waits Capture  ---------- ---------- -------------------- ---------- ----- ------------ -------  CONTENT\_OW LOB\_8K\_TS MZ\_ASSET\_WORK\_EVENT\_ INDEX 95 30.16  CONTENT\_OW LOB\_8K\_TS MZ\_CS\_WORK\_NOTE\_RE\_I \_2013\_1\_36 INDEX 48 15.24  CONTENT\_OW LOB\_8K\_TS MZ\_CS\_WORK\_INFO\_PART \_2013\_5\_35 INDEX 21 6.67  CONTENT\_OW INDEX\_TS MZ\_SALABLE\_FIRST\_AVA INDEX 21 6.67  CONTENT\_OW DATA\_TS MZ\_CS\_WORK\_PENDING\_R TABLE 20 6.35 |

关于 ITL的介绍详见： <http://www.askmaclean.com/archives/enqueue-tx-row-lock-index-itl-wait-event.html>

ITL Waits 等待 ITL 的次数，数据来源为 dba\_hist\_seg\_stat.itl\_waits\_delta

## 11-14   Segments by Buffer Busy Waits

|  |
| --- |
| Segments by Buffer Busy Waits DB/Inst: MAC/MAC2 Snaps: 70719-70723  -> % of Capture shows % of Buffer Busy Waits for each top segment compared  -> with total Buffer Busy Waits for all segments captured by the Snapshot  Buffer  Tablespace Subobject Obj. Busy % of  Owner Name Object Name Name Type Waits Capture  ---------- ---------- -------------------- ---------- ----- ------------ -------  CONTENT\_OW LOB\_8K\_TS MZ\_ASSET\_WORK\_EVENT\_ INDEX 251,073 57.07  CONTENT\_OW LOB\_8K\_TS MZ\_CS\_WORK\_NOTE\_RE\_I \_2013\_1\_36 INDEX 36,186 8.23  CONTENT\_OW LOB\_8K\_TS MZ\_CS\_WORK\_INFO\_PART \_2013\_5\_35 INDEX 31,786 7.23  CONTENT\_OW INDEX\_TS MZ\_AM\_REQUEST\_IX3A INDEX 15,663 3.56  CONTENT\_OW INDEX\_TS MZ\_CS\_WORK\_PENDING\_R INDEX 11,087 2.52 |

Buffer Busy Waits  该数据段上发生 buffer busy wait的次数   数据来源 dba\_hist\_seg\_stat.buffer\_busy\_waits\_delta

## 11-15   Segments by Global Cache Buffer

|  |
| --- |
| Segments by Global Cache Buffer BusyDB/Inst: MAC/MAC2 Snaps: 70719-7072  -> % of Capture shows % of GC Buffer Busy for each top segment compared  -> with GC Buffer Busy for all segments captured by the Snapshot  GC  Tablespace Subobject Obj. Buffer % of  Owner Name Object Name Name Type Busy Capture  ---------- ---------- -------------------- ---------- ----- ------------ -------  CONTENT\_OW INDEX\_TS MZ\_AM\_REQUEST\_IX3 INDEX 2,135,528 50.07  CONTENT\_OW DATA\_TS MZ\_CONTENT\_PROVIDER\_ TABLE 652,900 15.31  CONTENT\_OW LOB\_8K\_TS MZ\_ASSET\_WORK\_EVENT\_ INDEX 552,161 12.95  CONTENT\_OW LOB\_8K\_TS MZ\_CS\_WORK\_NOTE\_RE\_I \_2013\_1\_36 INDEX 113,042 2.65  CONTENT\_OW LOB\_8K\_TS MZ\_CS\_WORK\_INFO\_PART \_2013\_5\_35 INDEX 98,134 2.30 |

GC Buffer Busy 数据段上发挥僧gc buffer busy的次数， 数据源 dba\_hist\_seg\_stat.gc\_buffer\_busy\_delta

## 11-15   Segments by CR Blocks Received

|  |
| --- |
| Segments by CR Blocks Received DB/Inst: MAC/MAC2 Snaps: 70719-70723  -> Total CR Blocks Received: 763,037  -> Captured Segments account for 40.9% of Total  CR  Tablespace Subobject Obj. Blocks  Owner Name Object Name Name Type Received %Total  ---------- ---------- -------------------- ---------- ----- ------------ -------  CONTENT\_OW DATA\_TS MZ\_AM\_REQUEST TABLE 69,100 9.06  CONTENT\_OW DATA\_TS MZ\_CS\_WORK\_PENDING\_R TABLE 44,491 5.83  CONTENT\_OW INDEX\_TS MZ\_AM\_REQUEST\_IX3A INDEX 36,830 4.83  CONTENT\_OW DATA\_TS MZ\_PODCAST TABLE 36,632 4.80  CONTENT\_OW INDEX\_TS MZ\_AM\_REQUEST\_PK INDEX 19,646 2.57 |

CR  Blocks Received ：是指RAC中本地节点接收到global cache CR blocks 的数量； 数据来源为  dba\_hist\_seg\_stat.gc\_cu\_blocks\_received\_delta

%Total :   (该段上在本节点接收的Global CR blocks  / gc cr blocks received )

## 11-16     Segments by Current Blocks Received

|  |
| --- |
| Segments by Current Blocks ReceivedDB/Inst: MAC/MAC2 Snaps: 70719-70723  -> Total Current Blocks Received: 704,731  -> Captured Segments account for 61.8% of Total  Current  Tablespace Subobject Obj. Blocks  Owner Name Object Name Name Type Received %Total  ---------- ---------- -------------------- ---------- ----- ------------ -------  CONTENT\_OW INDEX\_TS MZ\_AM\_REQUEST\_IX3 INDEX 56,287 7.99  CONTENT\_OW INDEX\_TS MZ\_AM\_REQUEST\_IX3A INDEX 45,139 6.41  CONTENT\_OW DATA\_TS MZ\_AM\_REQUEST TABLE 40,350 5.73  CONTENT\_OW DATA\_TS MZ\_CS\_WORK\_PENDING\_R TABLE 22,808 3.24  CONTENT\_OW INDEX\_TS MZ\_AM\_REQUEST\_IX8 INDEX 13,343 1.89 |

Current  Blocks Received :是指RAC中本地节点接收到global cache Current blocks 的数量 ，数据来源DBA\_HIST\_SEG\_STAT.gc\_cu\_blocks\_received\_delta

%Total :   (该段上在本节点接收的 global cache current blocks / gc current blocks received)

# 12 Dictionary Cache Stats

|  |
| --- |
| Dictionary Cache Stats DB/Inst: MAC/MAC2 Snaps: 70719-70723  -> "Pct Misses" should be very low (< 2% in most cases) -> "Final Usage" is the number of cache entries being used  Get Pct Scan Pct Mod Final  Cache Requests Miss Reqs Miss Reqs Usage  ------------------------- ------------ ------ ------- ----- -------- ----------  dc\_awr\_control 87 2.3 0 N/A 6 1  dc\_global\_oids 1,134 7.8 0 N/A 0 13  dc\_histogram\_data 6,119,027 0.9 0 N/A 0 11,784  dc\_histogram\_defs 1,898,714 2.3 0 N/A 0 5,462  dc\_object\_grants 175 26.9 0 N/A 0 4  dc\_objects 10,254,514 0.2 0 N/A 0 3,807  dc\_profiles 8,452 0.0 0 N/A 0 2  dc\_rollback\_segments 3,031,044 0.0 0 N/A 0 1,947  dc\_segments 1,812,243 1.4 0 N/A 10 3,595  dc\_sequences 15,783 69.6 0 N/A 15,782 20  dc\_table\_scns 70 2.9 0 N/A 0 1  dc\_tablespaces 1,628,112 0.0 0 N/A 0 37  dc\_users 2,037,138 0.0 0 N/A 0 52  global database name 7,698 0.0 0 N/A 0 1  outstanding\_alerts 264 99.6 0 N/A 8 1  sch\_lj\_oids 51 7.8 0 N/A 0 1 |

Dictionary Cache 字典缓存也叫row cache

数据来源为dba\_hist\_rowcache\_summary

Cache  字典缓存类名kqrstcid <=> kqrsttxt  cid=3(dc\_rollback\_segments)

Get Requests  申请获取该数据字典缓存对象的次数     gets

Miss : GETMISSES 申请获取该数据字典缓存对象但 miss的次数

Pct Miss   ：　GETMISSES /Gets ， Miss的比例 ，这个pct miss应当非常低 小于2%，否则有出现大量row cache lock的可能

Scan Reqs：扫描申请的次数 ，kqrssc 、kqrpScan 、kqrpsiv时发生scan 会导致扫描数增加 kqrstsrq++(scan requests) ，例如migrate tablespace 时调用 kttm2b函数 为了安全删除uet$中的记录会callback kqrpsiv (used extent cache)，实际很少见

Pct Miss：SCANMISSES/SCANS

Mod Reqs:  申请修改字典缓存对象的次数，从上面的数据可以看到dc\_sequences的mod reqs很高，这是因为sequence是变化较多的字典对象

Final Usage  ：包含有有效数据的字典缓存记录的总数   也就是正在被使用的row cache记录 USAGE  Number of cache entries that contain valid data

|  |
| --- |
| Dictionary Cache Stats (RAC) DB/Inst: MAC/MAC2 Snaps: 70719-70723  GES GES GES  Cache Requests Conflicts Releases  ------------------------- ------------ ------------ ------------  dc\_awr\_control 14 2 0  dc\_global\_oids 88 0 102  dc\_histogram\_defs 43,518 0 43,521  dc\_objects 21,608 17 21,176  dc\_profiles 1 0 1  dc\_segments 24,974 14 24,428  dc\_sequences 25,178 10,644 347  dc\_table\_scns 2 0 2  dc\_tablespaces 165 0 166  dc\_users 119 0 119  outstanding\_alerts 478 8 250  sch\_lj\_oids 4 0 4 |

GES Request kqrstilr  total instance lock requests ，通过全局队列服务GES 来申请instance lock的次数

GES request 申请的原因可能是 dump cache object、kqrbfr LCK进程要background free some parent objects释放一些parent objects 等

GES Conflicts kqrstifr instance lock forced-releases     ， LCK进程以AST方式 释放锁的次数 ，仅出现在kqrbrl中

GES Releases  kqrstisr  instance lock self-releases ，LCK进程要background free some parent objects释放一些parent objects 时可能自增

上述数据中可以看到仅有dc\_sequences  对应的GES Conflicts较多， 对于sequence  使用ordered和non-cache选项会导致RAC中的一个边际效应，即”row cache lock”等待源于DC\_SEQUENCES ROW CACHE。 DC\_SEQUENCES 上的GETS request、modifications 、GES requests和GES conflict 与引发生成一个新的 sequence number的特定SQL执行频率相关。

在Oracle 10g中，ORDERED Sequence还可能在高并发下造成大量DFS lock Handle 等待，由于bug 5209859

# 13   Library Cache Activity

|  |
| --- |
| Library Cache Activity DB/Inst: MAC/MAC2 Snaps: 70719-70723  -> "Pct Misses" should be very low  Get Pct Pin Pct Invali-  Namespace Requests Miss Requests Miss Reloads dations  --------------- ------------ ------ -------------- ------ ---------- --------  ACCOUNT\_STATUS 8,436 0.3 0 N/A 0 0  BODY 8,697 0.7 15,537 0.7 49 0  CLUSTER 317 4.7 321 4.7 0 0  DBLINK 9,212 0.1 0 N/A 0 0  EDITION 4,431 0.0 8,660 0.0 0 0  HINTSET OBJECT 1,027 9.5 1,027 14.4 0 0  INDEX 792 18.2 792 18.2 0 0  QUEUE 10 0.0 1,733 0.0 0 0  RULESET 0 N/A 8 87.5 7 0  SCHEMA 8,169 0.0 0 N/A 0 0  SQL AREA 533,409 4.8 -4,246,727,944 101.1 44,864 576  SQL AREA BUILD 71,500 65.5 0 N/A 0 0  SQL AREA STATS 41,008 90.3 41,008 90.3 1 0  TABLE/PROCEDURE 320,310 0.6 1,033,991 3.6 25,378 0  TRIGGER 847 0.0 38,442 0.3 110 0 |

NameSpace   library cache 的命名空间

GETS  Requests  该命名空间所包含对象的library cache lock被申请的次数

GETHITS  对象的 library cache handle 正好在内存中被找到的次数

Pct Misses : ( 1-  ( GETHITS /GETS  Requests)) \*100

Pin Requests   该命名空间所包含对象上pin被申请的次数

PINHITS           要pin的对象的heap metadata正好在shared pool中的次数

Pct Miss   ( 1-  ( PINHITS  /Pin Requests)) \*100

Reloads  指从object handle 被重建开始不是第一次PIN该对象的PIN ，且该次PIN要求对象从磁盘上读取加载的次数 ;Reloads值较高的情况 建议增大shared\_pool\_size

INVALIDATIONS   由于以来对象被修改导致该命名空间所包含对象被标记为无效的次数

|  |
| --- |
| Library Cache Activity (RAC) DB/Inst: MAC/MAC2 Snaps: 70719-70723  GES Lock GES Pin GES Pin GES Inval GES Invali-  Namespace Requests Requests Releases Requests dations  --------------- ------------ ------------ ------------ ----------- -----------  ACCOUNT\_STATUS 8,436 0 0 0 0  BODY 0 15,497 15,497 0 0  CLUSTER 321 321 321 0 0  DBLINK 9,212 0 0 0 0  EDITION 4,431 4,431 4,431 0 0  HINTSET OBJECT 1,027 1,027 1,027 0 0  INDEX 792 792 792 0 0  QUEUE 8 1,733 1,733 0 0  RULESET 0 8 8 0 0  SCHEMA 4,226 0 0 0 0  TABLE/PROCEDURE 373,163 704,816 704,816 0 0  TRIGGER 0 38,430 38,430 0 0 |

GES Lock Request: dlm\_lock\_requests   Lock instance-lock ReQuests      申请获得lock instance lock的次数

GES PIN request : DLM\_PIN\_REQUESTS Pin instance-lock ReQuests   申请获得pin instance lock的次数

GES Pin Releases DLM\_PIN\_RELEASES release the pin instance lock     释放pin instance lock的次数

GES Inval Requests    DLM\_INVALIDATION\_REQUESTS  get the invalidation instance lock   申请获得invalidation instance lock的次数

GES Invali- dations    DLM\_INVALIDATIONS    接收到其他节点的invalidation pings次数

# 14 Process Memory Summary

|  |
| --- |
| Process Memory Summary DB/Inst: MAC/MAC2 Snaps: 70719-70723  -> B: Begin Snap E: End Snap  -> All rows below contain absolute values (i.e. not diffed over the interval)  -> Max Alloc is Maximum PGA Allocation size at snapshot time  -> Hist Max Alloc is the Historical Max Allocation for still-connected processes  -> ordered by Begin/End snapshot, Alloc (MB) desc  Hist  Avg Std Dev Max Max  Alloc Used Alloc Alloc Alloc Alloc Num Num  Category (MB) (MB) (MB) (MB) (MB) (MB) Proc Alloc  - -------- --------- --------- -------- -------- ------- ------- ------ ------  B Other 16,062.7 N/A 6.1 66.6 3,370 3,370 2,612 2,612  SQL 5,412.2 4,462.9 2.2 89.5 4,483 4,483 2,508 2,498  Freeable 2,116.4 .0 .9 6.3 298 N/A 2,266 2,266  PL/SQL 94.0 69.8 .0 .0 1 1 2,610 2,609  E Other 15,977.3 N/A 6.1 66.9 3,387 3,387 2,616 2,616  SQL 5,447.9 4,519.0 2.2 89.8 4,505 4,505 2,514 2,503  Freeable 2,119.9 .0 .9 6.3 297 N/A 2,273 2,273  PL/SQL 93.2 69.2 .0 .0 1 1 2,614 2,613 |

数据来源为dba\_hist\_process\_mem\_summary， 这里是对PGA 使用的一个小结，帮助我们了解到底谁用掉了PGA

B: 开始快照     E:  结束快照

该环节列出 PGA中各分类的使用量

Category   分类名，包括”SQL”, “PL/SQL”, “OLAP” 和”JAVA”. 特殊分类是 “Freeable” 和”Other”.    Free memory是指哪些 OS已经分配给进程，但没有分配给任何分类的内存。 “Other”是已经分配给分类的内存，但不是已命名的分类

Alloc (MB)  allocated\_total  该分类被分配的总内存

Used (MB)  used\_total  该分类已使用的内存

Avg  Alloc (MB) allocated\_avg     平均每个进程中该分类分配的内存量

Std Dev Alloc (MB) ：该分类分配的内存在每个进程之间的标准差

Max　Alloc　(MB) ALLOCATED\_MAX　：在快照时间内单个进程该分类最大分配过的内存量：Max Alloc is Maximum PGA Allocation size at snapshot time

Hist Max Alloc (MB) MAX\_ALLOCATED\_MAX: 目前仍链接着的进程该分类最大分配过的内存量：Hist Max Alloc is the Historical Max Allocation for still-connected processes

Num Proc num\_processes   进程数目

Num Alloc NON\_ZERO\_ALLOCS  分配了该类型 内存的进程数目

# 14 SGA信息

## 14 -1 SGA Memory Summary

|  |
| --- |
| SGA Memory Summary DB/Inst: MAC/MAC2 Snaps: 70719-70723  End Size (Bytes)  SGA regions Begin Size (Bytes) (if different)  ------------------------------ ------------------- -------------------  Database Buffers 20,669,530,112  Fixed Size 2,241,880  Redo Buffers 125,669,376  Variable Size 10,536,094,376  -------------------  sum 31,333,535,744 |

粗粒度的sga区域内存使用信息， End Size仅在于begin size不同时打印

## 14-2 SGA breakdown difference

|  |
| --- |
| SGA breakdown difference DB/Inst: MAC/MAC2 Snaps: 70719-70723  -> ordered by Pool, Name  -> N/A value for Begin MB or End MB indicates the size of that Pool/Name was  insignificant, or zero in that snapshot  Pool Name Begin MB End MB % Diff  ------ ------------------------------ -------------- -------------- -------  java free memory 64.0 64.0 0.00  large PX msg pool 7.8 7.8 0.00  large free memory 247.8 247.8 0.00  shared Checkpoint queue 140.6 140.6 0.00  shared FileOpenBlock 2,459.2 2,459.2 0.00  shared KGH: NO ACCESS 1,629.6 1,629.6 0.00  shared KGLH0 997.7 990.5 -0.71  shared KKSSP 312.2 308.9 -1.06  shared SQLA 376.6 370.6 -1.61  shared db\_block\_hash\_buckets 178.0 178.0 0.00  shared dbktb: trace buffer 156.3 156.3 0.00  shared event statistics per sess 187.1 187.1 0.00  shared free memory 1,208.9 1,220.6 0.97  shared gcs resources 435.0 435.0 0.00  shared gcs shadows 320.6 320.6 0.00  shared ges enqueues 228.9 228.9 0.00  shared ges resource 118.3 118.3 0.00  shared init\_heap\_kfsg 1,063.6 1,068.1 0.43  shared kglsim object batch 124.3 124.3 0.00  shared ksunfy : SSO free list 174.7 174.7 0.00  stream free memory 128.0 128.0 0.00  buffer\_cache 19,712.0 19,712.0 0.00  fixed\_sga 2.1 2.1 0.00  log\_buffer 119.8 119.8 0.00  ------------------------------------------------------------- |

Pool  内存池的名字

Name  内存池中细分组件的名字  例如KGLH0 存放KEL Heap 0 、SQLA存放SQL执行计划等

Begin MB 快照开始时该组件的内存大小

End MB  快照结束时该组件的内存大小

% Diff 差异百分比

特别注意 由于AMM /ASMM引起的shared pool收缩 一般在sga breakdown中可以提现 例如SQLA 、KQR等组件大幅缩小 ，可能导致一系列的解析等待 cursor: Pin S on X 、row cache lock等

此处的free memory信息也值得我们关注， 一般推荐shared pool应当有300~400  MB 的free memory为宜

# 15 Streams统计

|  |
| --- |
| Streams CPU/IO Usage DB/Inst: ORCL/orcl1 Snaps: 556-559  -> Streams processes ordered by CPU usage  -> CPU and I/O Time in micro seconds  Session Type CPU Time User I/O Time Sys I/O Time  ------------------------- -------------- -------------- --------------  QMON Coordinator 101,698 0 0  QMON Slaves 63,856 0 0  -------------------------------------------------------------  Streams Capture                           DB/Inst: CATGT/catgt  Snaps: 911-912  -> Lag Change should be small or negative (in seconds)                 Captured Enqueued      Pct      Pct      Pct     Pct                    Per      Per      Lag RuleEval  Enqueue RedoWait  Pause  Capture Name   Second   Second   Change     Time     Time      Time   Time  ------------ -------- -------- -------- -------- -------- -------- --------  CAPTURE\_CAT       650      391       93        0        23        0       71  -------------------------------------------------------------  Streams Apply                             DB/Inst: CATGT/catgt  Snaps: 911-912  -> Pct DB is the percentage of all DB transactions that this apply handled  -> WDEP is the wait for dependency  -> WCMT is the wait for commit  -> RBK is rollbacks -> MPS is messages per second  -> TPM is time per message in milli-seconds  -> Lag Change should be small or negative (in seconds)                  Applied  Pct  Pct  Pct Pct  Applied  Dequeue     Apply      Lag  Apply Name        TPS   DB WDEP WCMT RBK      MPS      TPM      TPM   Change  ------------ -------- ---- ---- ---- --- -------- -------- -------- --------  APPLY\_CAT           0     0     0    0    0        0        0         0         0            ------------------------------------------------------------- |

Capture Name : Streams捕获进程名

Captured Per Second ：每秒挖掘出来的message 条数

Enqueued Per Second:  每秒入队的message条数

lag change:  指日志生成的时间到挖掘到该日志生成 message的时间延迟

Pct Enqueue Time： 入队时间的比例

Pct redoWait  Time :  等待redo的时间比例

Pct Pause Time : Pause 时间的比例

Apply Name  Streams 应用Apply进程的名字

Applied TPS : 每秒应用的事务数

Pct DB:  所有的DB事务中 apply处理的比例

Pct WDEP: 由于等待依赖的数据而耗费的时间比例

Pct WCMT: 由于等待commit而耗费的时间比例

Pct RBK:  事务rollback 回滚的比例

Applied MPS: 每秒应用的message 数

Dequeue TPM: 每毫秒出队的message数

Lag Change:指最新message生成的时间到其被Apply收到的延迟

# 16 Resource Limit

|  |
| --- |
| Resource Limit Stats DB/Inst: MAC/MAC2 Snap: 70723  -> only rows with Current or Maximum Utilization > 80% of Limit are shown  -> ordered by resource name  Current Maximum Initial  Resource Name Utilization Utilization Allocation Limit  ------------------------------ ------------ ------------ ---------- ----------  ges\_procs 2,612 8,007 10003 10003  processes 2,615 8,011 10000 10000 |

数据源于dba\_hist\_resource\_limit

注意这里仅列出当前使用或最大使用量>80% \*最大限制的资源名，如果没有列在这里则说明 资源使用量安全  
Current Utilization 当前对该资源(包括Enqueue Resource、Lock和processes)的使用量

Maximum Utilization 从最近一次实例启动到现在该资源的最大使用量

Initial Allocation  初始分配值，一般等于参数文件中指定的值

Limit  实际上限值

# 17 init.ora Parameters

|  |
| --- |
| init.ora Parameters DB/Inst: MAC/MAC2 Snaps: 70719-70723  End value  Parameter Name Begin value (if different)  ----------------------------- --------------------------------- --------------  \_compression\_compatibility 11.2.0  \_kghdsidx\_count 4  \_ksmg\_granule\_size 67108864  \_shared\_pool\_reserved\_min\_all 4100  archive\_lag\_target 900  audit\_file\_dest /u01/app/oracle/admin/MAC/adum  audit\_trail OS  cluster\_database TRUE  compatible 11.2.0.2.0  control\_files +DATA/MAC/control01.ctl, +RECO  db\_16k\_cache\_size 268435456  db\_block\_size 8192  db\_cache\_size 19327352832  db\_create\_file\_dest +DATA |

Parameter Name 参数名

Begin value 开始快照时的参数值

End value 结束快照时的参数值 (仅在发生变化时打印)

# 18 Global Messaging Statistics

|  |
| --- |
| Global Messaging Statistics DB/Inst: MAC/MAC2 Snaps: 70719-70723  Statistic Total per Second per Trans  --------------------------------- ---------------- ------------ ------------  acks for commit broadcast(actual) 53,705 14.9 0.2  acks for commit broadcast(logical 311,182 86.1 1.3  broadcast msgs on commit(actual) 317,082 87.7 1.3  broadcast msgs on commit(logical) 317,082 87.7 1.3  broadcast msgs on commit(wasted) 263,332 72.9 1.1  dynamically allocated gcs resourc 0 0.0 0.0  dynamically allocated gcs shadows 0 0.0 0.0  flow control messages received 267 0.1 0.0  flow control messages sent 127 0.0 0.0  gcs apply delta 0 0.0 0.0  gcs assume cvt 55,541 15.4 0.2 |

全局通信统计信息，数据来源WRH$\_DLM\_MISC;

# 20 Global CR Served Stats

|  |
| --- |
| Global CR Served Stats DB/Inst: MAC/MAC2 Snaps: 70719-70723  Statistic Total  ------------------------------ ------------------  CR Block Requests 403,703  CURRENT Block Requests 444,896  Data Block Requests 403,705  Undo Block Requests 94,336  TX Block Requests 307,896  Current Results 652,746  Private results 21,057  Zero Results 104,720  Disk Read Results 69,418  Fail Results 508  Fairness Down Converts 102,844  Fairness Clears 15,207  Free GC Elements 0  Flushes 105,052  Flushes Queued 0  Flush Queue Full 0  Flush Max Time (us) 0  Light Works 71,793  Errors 117 |

LMS传输CR BLOCK的统计信息，数据来源WRH$\_CR\_BLOCK\_SERVER

# 21 Global CURRENT Served Stats

|  |
| --- |
| Global CURRENT Served Stats DB/Inst: MAC/MAC2 Snaps: 70719-70723  -> Pins = CURRENT Block Pin Operations  -> Flushes = Redo Flush before CURRENT Block Served Operations  -> Writes = CURRENT Block Fusion Write Operations  Statistic         Total   % <1ms  % <10ms % <100ms    % <1s   % <10s  ---------- ------------ -------- -------- -------- -------- --------  Pins             73,018    12.27    75.96     8.49     2.21     1.08  Flushes          79,336     5.98    50.17    14.45    19.45     9.95  Writes          102,189     3.14    35.23    19.34    33.26     9.03 |

数据来源dba\_hist\_current\_block\_server

Time to process current block request = (pin time + flush time + send time)

Pins CURRENT Block Pin Operations ， PIN的内涵是处理一个BAST  不包含对global current block的flush和实际传输

The pin time represents how much time is required to process a BAST. It does not include the flush time and  
the send time. The average pin time per block served should be very low because the processing consists  
mainly of code path and should never be blocked.

Flush 指 脏块被LMS进程传输出去之前，其相关的redo必须由LGWR已经flush 到磁盘上

Write 指fusion write number of writes which were mediated； 节点之间写脏块需求相互促成的行为 KJBL.KJBLREQWRITE  gcs write request msgs 、gcs writes refused

% <1ms  % <10ms % <100ms    % <1s   % <10s  分别对应为pin、flush、write行为耗时的比例

例如在上例中flush和 write 在1s 到10s之间的有9%，在100ms 和1s之间的有19%和33%，因为flush和write都是IO操作 所以这里可以预见IO存在问题，延迟较高

# 22 Global Cache Transfer Stats

|  |
| --- |
| Global Cache Transfer Stats DB/Inst: MAC/MAC2 Snaps: 70719-70723  -> Immediate (Immed) - Block Transfer NOT impacted by Remote Processing Delays  -> Busy (Busy) - Block Transfer impacted by Remote Contention  -> Congested (Congst) - Block Transfer impacted by Remote System Load  -> ordered by CR + Current Blocks Received desc  CR Current  ----------------------------- -----------------------------  Inst Block Blocks % % % Blocks % % %  No Class Received Immed Busy Congst Received Immed Busy Congst  ---- ----------- -------- ------ ------ ------ -------- ------ ------ ------  1 data block 133,187 76.3 22.6 1.1 233,138 75.2 23.0 1.7  4 data block 143,165 74.1 24.9 1.0 213,204 76.6 21.8 1.6  3 data block 122,761 75.9 23.0 1.1 220,023 77.7 21.0 1.3  1 undo header 104,219 95.7 3.2 1.1 941 93.4 5.8 .7  4 undo header 95,823 95.2 3.7 1.1 809 93.4 5.3 1.2  3 undo header 95,592 95.6 3.3 1.1 912 94.6 4.5 .9  1 undo block 25,002 95.8 3.4 .9 0 N/A N/A N/A  4 undo block 23,303 96.0 3.1 .9 0 N/A N/A N/A  3 undo block 21,672 95.4 3.7 .9 0 N/A N/A N/A  1 Others 1,909 92.0 6.8 1.2 6,057 89.6 8.9 1.5  4 Others 1,736 92.4 6.1 1.5 5,841 88.8 9.9 1.3  3 Others 1,500 92.4 5.9 1.7 4,405 87.7 10.8 1.6 |

数据来源DBA\_HIST\_INST\_CACHE\_TRANSFER

Inst No 节点号

Block Class 块的类型

CR Blocks Received 该节点上 该类型CR 块的接收数量

CR Immed %: CR块请求立即接收到的比例

CR Busy%：CR块请求由于远端争用而没有立即接收到的比例

CR Congst%: CR块请求由于远端负载高而没有立即接收到的比例

Current Blocks Received  该节点上 该类型Current 块的接收数量

Current Immed %: Current块请求立即接收到的比例

Current Busy%：Current块请求由于远端争用而没有立即接收到的比例

Current Congst%: Current块请求由于远端负载高而没有立即接收到的比例

Congst%的比例应当非常低 不高于2%， Busy%很大程度受到IO的影响，如果超过10% 一般会有严重的gc buffer busy acquire/release

补充 RAC 相关指标 内容由 tong.wang@parnassusdata.com 整理

# RAC相关指标

## Global Cache Load Profile

|  |  |  |
| --- | --- | --- |
|  | Per Second | Per Transaction |
| Global Cache blocks received: | 12.06 | 2.23 |
| Global Cache blocks served: | 8.18 | 1.51 |
| GCS/GES messages received: | 391.19 | 72.37 |
| GCS/GES messages sent: | 368.76 | 68.22 |
| DBWR Fusion writes: | 0.10 | 0.02 |
| Estd Interconnect traffic (KB) | 310.31 |  |

|  |  |
| --- | --- |
| 指标 | 指标说明 |
| Global Cache blocks received | 通过硬件连接收到远程实例的数据块的数量。发生在一个进程请求一致性读一个数据块不是在本地缓存中。Oracle发送一个请求到另外的实例。一旦缓冲区收到，这个统计值就会增加。这个统计值是另两个统计值的和：Global Cache blocks received = gc current blocks received + gc cr blocks received |
| Global Cache blocks served | 通过硬件连接发送到远程实例的数据块的数量。这个统计值是另外两个统计值的和：Global Cache blocks served = gc current blocks served + gc cr blocks served |
| GCS/GES messages received | 通过硬件连接收到远程实例的消息的数量。这个统计值通常代表RAC服务引起的开销。这个统计值是另外两个统计值的和：GCS/GES messages received = gcs msgs received + ges msgs received |
| GCS/GES messages sent | 通过硬件连接发送到远程实例的消息的数量。这个统计值通常代表RAC服务引起的开销。这个统计值是另外两个统计值的和：GCS/GES messages sent = gcs messages sent + ges messages sent |
| DBWR Fusion writes | 这个统计值显示融合写入的次数。在RAC中，单实例Oracle数据库，数据块只被写入磁盘因为数据过期，缓冲替换或者发生检查点。当一个数据块在缓存中被替换因为数据过期或发生检查点但在另外的实例没有写入磁盘，Global Cache Service会请求实例将数据块写入磁盘。因此融合写入不包括在第一个实例中的额外写入磁盘。大量的融合写入表明一个持续的问题。实例产生的融合写入请求占总的写入请求的比率用于性能分析。高比率表明DB cache大小不合适或者检查点效率低。 |
| Estd Interconnect traffic (KB) | 连接传输的KB大小。计算公式如下：Estd Interconnect traffic (KB) = ((‘gc cr blocks received’+ ‘gc current blocks received’ + ‘gc cr blocksserved’+ ‘gc current blocks served’) \* Block size)  + ((‘gcs messages sent’ + ‘ges messages sent’ + ‘gcs msgs received’+ ‘gcs msgs  received’)\*200)/1024/Elapsed Time |

## Global Cache Efficiency Percentages (Target local+remote 100%)

|  |  |
| --- | --- |
| Buffer access – local cache %: | 91.05 |
| Buffer access – remote cache %: | 0.03 |
| Buffer access – disk %: | 8.92 |

|  |  |
| --- | --- |
| 指标 | 指标说明 |
| Buffer access – local cache % | 数据块从本地缓存命中占会话总的数据库请求次数的比例。在OLTP应用中最希望的是尽可能维持这个比率较高，因为这是最低成本和最快速的获得数据库数据块的方法。计算公式：Local Cache Buffer Access Ratio = 1 – ( physical reads cache + Global Cache blocks received ) / Logical Reads |
| Buffer access – remote cache % | 数据块从远程实例缓存命中占会话总的数据块请求的比例。在OLTP应用中这个比率和Buffer access – local cache的和应该尽可能的高因为这两种方法访问数据库数据块是最快速最低成本的。这个比率的计算方法：Remote Cache Buffer Access Ratio = Global Cache blocks received / Logical Reads |
| Buffer access – disk % | 从磁盘上读数据块到缓存占会话总的数据块请求次数的比例。在OLTP应用中希望维持这个比例低因为物理读是最慢的访问数据库数据块的方式。这个比率计算方法：1 – physical reads cache / Logical Reads |

## Global Cache and Enqueue Services – Workload Characteristics

|  |  |
| --- | --- |
| Avg global enqueue get time (ms): | 0.0 |
| Avg global cache cr block receive time (ms): | 0.3 |
| Avg global cache current block receive time (ms): | 0.2 |
| Avg global cache cr block build time (ms): | 0.0 |
| Avg global cache cr block send time (ms): | 0.0 |
| Global cache log flushes for cr blocks served %: | 1.2 |
| Avg global cache cr block flush time (ms): | 1.8 |
| Avg global cache current block pin time (ms): | 1,021.7 |
| Avg global cache current block send time (ms): | 0.0 |
| Global cache log flushes for current blocks served %: | 6.9 |
| Avg global cache current block flush time (ms): | 0.9 |

本文永久地址： http://www.askmaclean.com/archives/rac-awr-statistics.html

|  |  |
| --- | --- |
| 指标 | 指标说明 |
| Avg global enqueue get time (ms) | 通过interconnect发送消息，为争夺资源开启一个新的全局队列或者对已经开启的队列转换访问模式所花费的时间。如果大于20ms，你的系统可能会出现超时。 |
| Avg global cache cr block receive time (ms) | 从请求实例发送消息到mastering instance（2-way get）和一些到holding instance (3-way get)花费的时间。这个时间包括在holding instance生成数据块一致性读映像的时间。CR数据块获取耗费的时间不应该大于15ms。 |
| Avg global cache current block receive time (ms) | 从请求实例发送消息到mastering instance（2-way get）和一些到holding instance (3-way get)花费的时间。这个时间包括holding instance日志刷新花费的时间。Current Block获取耗费的时间不大于30ms |
| Avg global cache cr block build time (ms) | CR数据块创建耗费的时间 |
| Avg global cache cr block send time (ms) | CR数据块发送耗费的时间 |
| Global cache log flushes for cr blocks served % | 需要日志刷新的CR数据块占总的需要服务的CR数据块的比例。 |
| Avg global cache cr block flush time (ms) | CR数据块刷新耗费的时间 |
| Avg global cache current block pin time (ms) | Current数据块pin耗费的时间 |
| Avg global cache current block send time (ms) | Current数据块发送耗费的时间 |
| Global cache log flushes for current blocks served % | 需要日志刷新的Current数据块占总的需要服务的Current数据块的比例 |
| Avg global cache current block flush time (ms) | Current数据块刷新耗费的时间 |

## Global Cache and Enqueue Services – Messaging Statistics

|  |  |
| --- | --- |
| Avg message sent queue time (ms): | 2,367.6 |
| Avg message sent queue time on ksxp (ms): | 0.1 |
| Avg message received queue time (ms): | 0.3 |
| Avg GCS message process time (ms): | 0.0 |
| Avg GES message process time (ms): | 0.0 |
| % of direct sent messages: | 54.00 |
| % of indirect sent messages: | 44.96 |
| % of flow controlled messages: | 1.03 |

|  |  |
| --- | --- |
| 指标 | 指标说明 |
| Avg message sent queue time (ms) | 一条信息进入队列到发送它的时间 |
| Avg message sent queue time on ksxp (ms) | 对端收到该信息并返回ACK的时间，这个指标很重要，直接反应了网络延迟，一般小于1ms |
| Avg message received queue time (ms) | 一条信息进入队列到收到它的时间 |
| Avg GCS message process time (ms) |  |
| Avg GES message process time (ms) |  |
| % of direct sent messages | 直接发送信息占的比率 |
| % of indirect sent messages | 间接发送信息占的比率，一般是排序或大的信息，流控制也可能引起 |
| % of flow controlled messages | 流控制信息占的比率，流控制最常见的原因是网络状况不佳， % of flowcontrolled messages应当小于1% |

## Wait Event Histogram

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | % of Waits | | | | | | | |
| Event | Total Waits | <1ms | <2ms | <4ms | <8ms | <16ms | <32ms | <=1s | >1s |
| ADR block file read | 208 | 38.0 |  | 3.4 | 44.7 | 13.9 |  |  |  |
| ADR block file write | 40 | 100.0 |  |  |  |  |  |  |  |
| ADR file lock | 48 | 100.0 |  |  |  |  |  |  |  |
| ARCH wait for archivelog lock | 3 | 100.0 |  |  |  |  |  |  |  |
| ASM file metadata operation | 12.8K | 99.7 | .1 | .0 |  | .0 | .0 | .2 | .0 |
| Backup: MML write backup piece | 310.5K | 7.6 | .1 | .1 | 1.3 | 10.4 | 30.2 | 50.2 | .0 |
| CGS wait for IPC msg | 141.7K | 100.0 |  |  |  |  |  |  |  |
| CSS initialization | 34 | 50.0 |  |  | 47.1 | 2.9 |  |  |  |
| CSS operation: action | 110 | 48.2 | 20.9 | 28.2 | 2.7 |  |  |  |  |
| CSS operation: query | 102 | 88.2 | 3.9 | 7.8 |  |  |  |  |  |
| DFS lock handle | 6607 | 93.9 | .5 | .2 | .0 |  | .0 | 5.3 | .0 |
| Disk file operations I/O | 1474 | 100.0 |  |  |  |  |  |  |  |
| IPC send completion sync | 21.9K | 99.5 | .1 | .1 | .1 | .0 | .2 |  |  |
| KJC: Wait for msg sends to complete | 13 | 100.0 |  |  |  |  |  |  |  |
| LGWR wait for redo copy | 16.3K | 100.0 | .0 |  |  |  |  |  |  |
| Log archive I/O | 3 | 33.3 | 66.7 |  |  |  |  |  |  |
| PX Deq: Signal ACK EXT | 2256 | 99.8 | .1 | .1 |  |  |  |  |  |
| PX Deq: Signal ACK RSG | 2124 | 99.9 | .1 | .0 |  |  |  |  |  |
| PX Deq: Slave Session Stats | 7997 | 94.6 | .9 | .9 | 2.5 | .8 | .4 |  |  |
| PX Deq: Table Q qref | 2355 | 99.9 | .1 |  |  |  |  |  |  |
| PX Deq: reap credit | 1215.7K | 100.0 | .0 | .0 |  |  |  |  |  |
| PX qref latch | 1366 | 100.0 |  |  |  |  |  |  |  |
| Parameter File I/O | 194 | 94.8 | 1.0 |  | 1.0 | 1.0 |  | 1.5 | .5 |

Wait Event Histogram：等待时间直方图

Event：等待事件名字

Total Waits：该等待事件在快照时间内等待的次数

%of Waits < 1ms ：小于1ms的等待次数

%of Waits < 2ms ：小于2ms的等待次数

%of Waits < 4ms ：小于4ms的等待次数

%of Waits < 8ms ：小于8ms的等待次数

%of Waits < 16ms ：小于16ms的等待次数

%of Waits < 32ms ：小于32ms的等待次数

%of Waits < =1s ：小于等于1s的等待次数

%of Waits > 1s ：大于1s的等待次数

## Parent Latch Statistics

only latches with sleeps are shown

ordered by name

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Latch Name | Get Requests | Misses | Sleeps | Spin & Sleeps 1->3+ |
| Real-time plan statistics latch | 77,840 | 136 | 20 | 116/0/0/0 |
| active checkpoint queue latch | 321,023 | 20,528 | 77 | 20451/0/0/0 |
| active service list | 339,641 | 546 | 132 | 424/0/0/0 |
| call allocation | 328,283 | 550 | 148 | 440/0/0/0 |
| enqueues | 1,503,525 | 217 | 14 | 203/0/0/0 |
| ksuosstats global area | 2,605 | 1 | 1 | 0/0/0/0 |
| messages | 2,608,863 | 141,380 | 29 | 141351/0/0/0 |
| name-service request queue | 155,047 | 43 | 15 | 28/0/0/0 |
| qmn task queue latch | 2,368 | 90 | 78 | 12/0/0/0 |
| query server process | 268 | 30 | 30 | 0/0/0/0 |
| redo writing | 910,703 | 11,623 | 50 | 11573/0/0/0 |
| resmgr:free threads list | 14,454 | 190 | 4 | 186/0/0/0 |
| space background task latch | 11,209 | 15 | 7 | 8/0/0/0 |

Latch Name：闩名称

Get Requests：申请获得父闩的次数

[本文永久地址http://www.askmaclean.com/archives/rac-awr-statistics.html](http://www.askmaclean.com/archives/rac-awr-statistics.html)

## Child Latch Statistics

only latches with sleeps/gets > 1/100000 are shown

ordered by name, gets desc

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Latch Name | Child Num | Get Requests | Misses | Sleeps | Spin & Sleeps 1->3+ |
| KJC message pool free list | 1 | 96,136 | 82 | 20 | 62/0/0/0 |
| Lsod array latch | 10 | 2,222 | 153 | 118 | 58/0/0/0 |
| Lsod array latch | 13 | 2,151 | 43 | 14 | 29/0/0/0 |
| Lsod array latch | 4 | 2,066 | 154 | 124 | 59/0/0/0 |
| Lsod array latch | 5 | 1,988 | 105 | 44 | 63/0/0/0 |
| Lsod array latch | 9 | 1,734 | 95 | 32 | 64/0/0/0 |
| Lsod array latch | 2 | 1,707 | 88 | 38 | 55/0/0/0 |
| Lsod array latch | 11 | 1,695 | 88 | 32 | 57/0/0/0 |
| Lsod array latch | 6 | 1,680 | 158 | 126 | 64/0/0/0 |
| Lsod array latch | 12 | 1,657 | 155 | 111 | 65/0/0/0 |
| Lsod array latch | 7 | 1,640 | 90 | 34 | 59/0/0/0 |
| Lsod array latch | 1 | 1,627 | 169 | 153 | 46/0/0/0 |
| Lsod array latch | 3 | 1,555 | 87 | 36 | 54/0/0/0 |
| Lsod array latch | 8 | 1,487 | 127 | 88 | 57/0/0/0 |
| cache buffers chains | 47418 | 354,313 | 391 | 4 | 387/0/0/0 |
| cache buffers chains | 8031 | 337,135 | 250 | 8 | 242/0/0/0 |
| cache buffers chains | 78358 | 305,022 | 528 | 9 | 519/0/0/0 |
| cache buffers chains | 6927 | 241,808 | 129 | 4 | 125/0/0/0 |

Latch Name：闩名称

Child Num：

Get Requests：

Misses：

Sleeps：

Spin&Sleeps 1->3+：

## Dictionary Cache Stats (RAC)

|  |  |  |  |
| --- | --- | --- | --- |
| Cache | GES Requests | GES Conflicts | GES Releases |
| dc\_awr\_control | 11 | 5 | 0 |
| dc\_global\_oids | 5 | 0 | 0 |
| dc\_histogram\_defs | 215 | 1 | 707 |
| dc\_objects | 90 | 9 | 0 |
| dc\_segments | 79 | 10 | 73 |
| dc\_sequences | 35,738 | 37 | 0 |
| dc\_table\_scns | 6 | 0 | 0 |
| dc\_tablespace\_quotas | 907 | 77 | 0 |
| dc\_users | 10 | 0 | 0 |
| outstanding\_alerts | 576 | 288 | 0 |

Cache：字典缓存类名

GES Requests：

GES Conflicts：

GES Releases：

## Library Cache Activity (RAC)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Namespace | GES Lock Requests | GES Pin Requests | GES Pin Releases | GES Inval Requests | GES Invali- dations |
| ACCOUNT\_STATUS | 242 | 0 | 0 | 0 | 0 |
| BODY | 0 | 1,530,013 | 1,530,013 | 0 | 0 |
| CLUSTER | 74 | 74 | 74 | 0 | 0 |
| DBLINK | 246 | 0 | 0 | 0 | 0 |
| EDITION | 311 | 311 | 311 | 0 | 0 |
| HINTSET OBJECT | 186 | 186 | 186 | 0 | 0 |
| INDEX | 152,360 | 152,360 | 152,360 | 0 | 0 |
| QUEUE | 223 | 9,717 | 9,717 | 0 | 0 |
| SCHEMA | 255 | 0 | 0 | 0 | 0 |
| SUBSCRIPTION | 0 | 26 | 26 | 0 | 0 |
| TABLE/PROCEDURE | 275,215 | 3,023,083 | 3,023,083 | 0 | 0 |
| TRIGGER | 0 | 384,493 | 384,493 | 0 | 0 |

Namespace：library cache 的命名空间

GES Lock Requests：

GES Pin Requests：

GES Inval Requests：

GES Invali-dations：

## Interconnect Ping Latency Stats

Ping latency of the roundtrip of a message from this instance to

target instances.

The target instance is identified by an instance number.

Average and standard deviation of ping latency is given in miliseconds

for message sizes of 500 bytes and 8K.

Note that latency of a message from the instance to itself is used as

control, since message latency can include wait for CPU

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Target Instance | 500B Ping Count | Avg Latency 500B msg | Stddev 500B msg | 8K Ping Count | Avg Latency 8K msg | Stddev 8K msg |
| 1 | 1,138 | 0.20 | 0.03 | 1,138 | 0.20 | 0.03 |
| 2 | 1,138 | 0.17 | 0.04 | 1,138 | 0.20 | 0.05 |
| 3 | 1,138 | 0.19 | 0.22 | 1,138 | 0.23 | 0.22 |
| 4 | 1,138 | 0.18 | 0.04 | 1,138 | 0.21 | 0.04 |

Target Instance：目标实例

500B Ping Count：

Avg Latency 500B msg：

Stddev 500B msg：

8K Ping Count：

Avg Latency 8K msg：

Stddev 8K msg：

## Interconnect Throughput by Client

Throughput of interconnect usage by major consumers

All throughput numbers are megabytes per second

|  |  |  |
| --- | --- | --- |
| Used By | Send Mbytes/sec | Receive Mbytes/sec |
| Global Cache | 0.10 | 0.20 |
| Parallel Query | 0.02 | 0.06 |
| DB Locks | 0.09 | 0.09 |
| DB Streams | 0.00 | 0.00 |
| Other | 0.02 | 0.01 |

Used By：主要消费者

Send Mbytes/sec：发送Mb/每秒

Receive Mbytes/sec：接收Mb/每秒

## Interconnect Device Statistics

Throughput and errors of interconnect devices (at OS level)

All throughput numbers are megabytes per second

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Device Name | IP Address | Public | Source | Send Mbytes/sec | Send Errors | Send Dropped | Send Buffer Overrun | Send Carrier Lost | Receive Mbytes/sec | Receive Errors | Receive Dropped | Receive Buffer Overrun | Receive Frame Errors |
| bondib0 | 192.168.10.8 | NO | cluster\_interconnects parameter | 0.00 | 0 | 0 | 0 | 0 | 0.00 | 0 | 0 | 0 |  |

Device Name：设备名称

IP Address：IP地址

Public：是否为公用网络

Source：来源

Send Mbytes/sec：发送MB/每秒

Send Errors：发送错误

Send Dropped：

Send Buffer Overrun：

Send Carrier Lost：

Receive Mbytes/sec：

Receive Errors：

Receive Dropped：

Receive Buffer Overrun：

Receive Frame Errors：

## Dynamic Remastering Stats

times are in seconds

Affinity objects – objects mastered due to affinity at begin/end snap

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Total | per Remaster Op | Begin Snap | End Snap |
| remaster ops | 29 | 1.00 |  |  |
| remastered objects | 40 | 1.38 |  |  |
| replayed locks received | 1,990 | 68.62 |  |  |
| replayed locks sent | 877 | 30.24 |  |  |
| resources cleaned | 0 | 0.00 |  |  |
| remaster time (s) | 5.0 | 0.17 |  |  |
| quiesce time (s) | 1.7 | 0.06 |  |  |
| freeze time (s) | 0.6 | 0.02 |  |  |
| cleanup time (s) | 0.7 | 0.02 |  |  |
| replay time (s) | 0.2 | 0.01 |  |  |
| fixwrite time (s) | 1.3 | 0.04 |  |  |
| sync time (s) | 0.5 | 0.02 |  |  |
| affinity objects |  |  | 365 | 367 |

Name：

Total：

Per Remaster Op：

Begin Snap：

End Snap：

## 参考文档

Statistics Descriptions  http://docs.oracle.com/cd/B19306\_01/server.102/b14237/stats002.htm

Memory Configuration and Use  http://docs.oracle.com/cd/B19306\_01/server.102/b14211/memory.htm

Library Cache Hit (%)   http://docs.oracle.com/cd/B16240\_01/doc/doc.102/e16282/oracle\_database\_help/oracle\_database\_instance\_efficiency\_libcache\_hit\_pct.html

Oracle® Database Performance Tuning Guide 12c Release 1 (12.1)

How to Interpret the “SQL ordered by Physical Reads (UnOptimized)” Section in AWR Reports (11.2 onwards) [ID 1466035.1]

# 常见问题:如何使用AWR报告来诊断数据库性能问题 (Doc ID 1523048.1)

## 适用于:

Oracle Database - Enterprise Edition - 版本 10.2.0.1 和更高版本

本文档所含信息适用于所有平台

## 目标

本文旨在提供如何解释跟数据库性能问题息息相关的AWR信息。

需要注意的是生成 AWR Report 或访问 AWR 相关的视图，以及使用任何 AWR 相关的诊断信息，都需要额外的 Diagnostic Pack License。这包括生成 AWR/ADDM/ASH report，也包括当技术支持要求的生成上述报表时。

注意: Oracle Diagnostics Pack (以及 Oracle Tuning Pack) 只在企业版中提供。

详见:

Oracle? Database Licensing Information

12c Release 1 (12.1)

Part number E17614-08

Chapter 1 1 Oracle Database Editions

Feature Availability by Edition

http://docs.oracle.com/cd/E16655\_01/license.121/e17614/editions.htm#DBLIC116

# 最佳实践

## 如何主动避免问题发生及做好诊断信息的收集

有些问题是无法预见的，但大部分其它的问题如果及早发现一些征兆其实是可以避免的。同时，如果问题确实发生了，那么收集问题发生时的信息就非常重要。有关于如何主动避免问题及诊断信息的收集，请参见：

Document 1482811.1 Best Practices: Proactively Avoiding Database and Query Performance Issues

Document 1477599.1 Best Practices Around Data Collection For Performance Issues

提出问题、获取帮助并分享您的经验

您想要与其他 Oracle 客户、Oracle 员工及业内专家深入探讨吗？

Click here to join the discussion where you can ask questions, get help from others, and share your experiences with this specific article.

点击这里访问 My Oracle Support Community 数据库性能优化页，在这里您可以提出问题、获取他人的帮助并分享您的经验。

## 解决方案

对于数据库整体的性能问题，AWR的报告是一个非常有用的诊断工具。

一般来说，当检测到性能问题时，我们会收集覆盖了发生问题的时间段的AWR报告-但是最好只收集覆盖1个小时时间段的AWR报告-如果时间过长，那么AWR报告就不能很好的反映出问题所在。

还应该收集一份没有性能问题的时间段的AWR报告，作为一个参照物来对比有问题的时间段的AWR报告。这两个AWR报告的时间段应该是一致的，比如都是半个小时的，或者都是一个小时的。

## 关于如何收集AWR报告，请参照如下文档：

Document 1363422.1 Automatic Workload Repository (AWR) Reports - Start Point

注：最好一开始我们从ADDM报告入手，因为对应时间段的ADDM报告往往已经指出了问题所在。

参见: Use of ADDM Reports alongside AWR

## Interpretation

在处理性能问题时，我们最关注的是数据库正在等待什么。

当进程因为某些原因不能进行操作时，它需要等待。花费时间最多的等待事件是我们最需要关注的，因为降低它，我们能够获得最大的好处。

AWR报告中的"Top 5 Timed Events"部分就提供了这样的信息，可以让我们只关注主要的问题。

## Top 5 Timed Events

正如前面提到的，"Top 5 Timed Events"是AWR报告中最重要的部分。它指出了数据库的sessions花费时间最多的等待事件，如下：

Top 5 Timed Events Avg %Total

~~~~~~~~~~~~~~~~~~ wait Call

Event Waits Time (s) (ms) Time Wait Class

------------------------------ ------------ ----------- ------ ------ ----------

db file scattered read 10,152,564 81,327 8 29.6 User I/O

db file sequential read 10,327,231 75,878 7 27.6 User I/O

CPU time 56,207 20.5

read by other session 4,397,330 33,455 8 12.2 User I/O

PX Deq Credit: send blkd 31,398 26,576 846 9.7 Other

-------------------------------------------------------------

Top 5 Events部分包含了一些跟Events（事件）相关的信息。它记录了这期间遇到的等待的总次数，等待所花费的总时间，每次等待的平均时间；这一部分是按照每个Event占总体call time的百分比来进行排序的。

根 据Top 5 Events部分的信息的不同，接下来我们需要检查AWR报告的其他部分，来验证发现的问题或者做定量分析。等待事件需要根据报告期的持续时间和当时数据 库中的并发用户数进行评估。如：10分钟内1000万次的等待事件比10个小时内的1000万等待更有问题；10个用户引起的1000万次的等待事件比 10,000个用户引起的相同的等待要更有问题。

就像上面的例子，将近60%的时间是在等待IO相关的事件。

事件"db file scattered read"一般表明正在做由全表扫描或者index fast full scan引起的多块读。

事件"db file sequential read"一般是由不能做多块读的操作引起的单块读（如读索引）

其他20%的时间是花在使用或等待CPU time上。过高的CPU使用经常是性能不佳的SQL引起的（或者这些SQL有可能用更少的资源完成同样的操作）；对于这样的SQL，过多的IO操作也是一个症状。关于CPU使用方面，我们会在之后讨论。

在以上基础上，我们将调查是否这个等待事件是有问题的。若有问题，解决它；若是正常的，检查下个等待事件。

过多的IO相关的等待一般会有两个主要的原因：

* 数据库做了太多的读操作
* 每次的IO读操作都很慢

Top 5 Events部分的显示的信息会帮助我们检查：

## 是否数据库做了大量的读操作：

上面的图显示了在这段时间里两类读操作都分别大于1000万，这些操作是否过多取决于报告的时间是1小时或1分钟。我们可以检查AWR报告的elapsed time

如果这些读操作确实是太多了，接下来我们需要检查AWR报告中 SQL Statistics 部分的信息，因为读操作都是由SQL语句发起的。

是否是每次的IO读操作都很慢：

上面的图显示了在这段时间里两类读操作平均的等待时间是小于8ms的

至于8ms是快还是慢取决于底层的硬件设备；一般来讲小于20ms的都可以认为是可以接受的。

我们还可以在AWR报告"Tablespace IO Stats"部分得到更详细的信息

Tablespace IO Stats DB/Inst: VMWREP/VMWREP Snaps: 1-15

-> ordered by IOs (Reads + Writes) desc

Tablespace

------------------------------

Av Av Av Av Buffer Av Buf

Reads Reads/s Rd(ms) Blks/Rd Writes Writes/s Waits Wt(ms)

-------------- ------- ------ ------- ------------ -------- ---------- ------

TS\_TX\_DATA

14,246,367 283 7.6 4.6 145,263,880 2,883 3,844,161 8.3

USER

204,834 4 10.7 1.0 17,849,021 354 15,249 9.8

UNDOTS1

19,725 0 3.0 1.0 10,064,086 200 1,964 4.9

AE\_TS

4,287,567 85 5.4 6.7 932 0 465,793 3.7

TEMP

2,022,883 40 0.0 5.8 878,049 17 0 0.0

UNDOTS3

1,310,493 26 4.6 1.0 941,675 19 43 0.0

TS\_TX\_IDX

1,884,478 37 7.3 1.0 23,695 0 73,703 8.3

>SYSAUX

346,094 7 5.6 3.9 112,744 2 0 0.0

SYSTEM

101,771 2 7.9 3.5 25,098 0 653 2.7

如上图，我们关心Av Rd(ms)的指标。如果它高于20ms并且同时有很多读操作的，我们可能要开始从OS的角度调查是否有潜在的IO问题。

注：对于一些比较空闲的tablespace/files,我们可能会得到一个比较大的Av Rd(ms)值；对于这样的情况，我们应该忽略这样的tablespace/files;因为这个很大的值可能是由于硬盘自旋(spin)引起的，没有太大的参考意义。比如对

于一个有1000万次读操作而且很慢的系统，引起问题的基本不可能是一个只有10次read的tablespace/file

以下的文档可以帮助我们进一步调查IO方面的问题：

Note:223117.1 Troubleshooting I/O-related waits

虽 然高"db file scattered read"和"db file sequential read"等待可以是I / O相关的问题，但是很多时候这些等待也可能是正常的；实际上，对一个已经性能很好的数据库系统，这些等待事件往往在top 5等待事件里，因为这意味着您的数据库没有那些真正的“问题”。

诀窍是能够评估引起这些等待的语句是否使用了最优的访问路径。如果"db file scattered read"比较高，那么相关的SQL语句可能使用了全表扫描而没有使用索引（也许是没有创建索引，也许是没有合适的索引）；相应的，如果"db file sequential read"过多，则表明也许是这些SQL语句使用了selectivity不高的索引从而导致访问了过多不必要的索引块或者使用了错误的索引。这些等待可 能说明SQL语句的执行计划不是最优的。

接下来就需要通过AWR来检查这些top SQL是否可以进一步的调优，我们可以查看AWR报告中 SQL Statistics 的部分.

上面的例子显示了20%的时间花在了等待或者使用CPU上，我们也需要检查 SQL statistics 部分来进一步的分析。

需要注意，接下来的分析步骤取决于我们在TOP 5部分的发现。在上面的例子里，3个top wait event表明问题可能与SQL语句执行计划不好有关，所以接下来我们要去分析"SQL Statistics"部分。

同样的，因为我们并没有看到latch相关的等待，latch在我们这个例子里并没有引发严重的性能问题；那么我们接下来就完全不需要分析latch相关的信息。

一 般来讲，如果数据库性能很慢，TOP 5等待事件里"CPU"， "db file sequential read" 和"db file scattered read" 比较明显（不管它们之间的顺序如何），我们总是需要检查Top SQL (by logical and physical reads)部分；调用SQL Tuning Advisor或者手工调优这些SQL来确保它们是有效率的运行。

SQL Statistics

AWR包含了一些不同的SQL统计值：

sql stats

根据Top 5 部分的Top Wait Event不同，我们需要检查不同的SQL statistic。

在我们这个例子里，Top Wait Event是"db file scattered read"，"db file sequential read"和CPU；我们最需要关心的是SQL ordered by CPU Time, Gets and Reads。

我们会从"SQL ordered by gets"入手，因为引起高buffer gets的SQL语句一般是需要调优的对象。

SQL ordered by Gets

-> Resources reported for PL/SQL code includes the resources used by all SQL

statements called by the code.

-> Total Buffer Gets: 4,745,943,815

-> Captured SQL account for 122.2% of Total

Gets CPU Elapsed

Buffer Gets Executions per Exec %Total Time (s) Time (s) SQL Id

-------------- ------------ ------------ ------ -------- --------- -------------

1,228,753,877 168 7,314,011.2 25.9 8022.46 8404.73 5t1y1nvmwp2

SELECT ADDRESSID",CURRENT$."ADDRESSTYPEID",CURRENT$URRENT$."ADDRESS3",

CURRENT$."CITY",CURRENT$."ZIP",CURRENT$."STATE",CURRENT$."PHONECOUNTRYCODE",

CURRENT$."PHONENUMBER",CURRENT$."PHONEEXTENSION",CURRENT$."FAXCOU

1,039,875,759 62,959,363 16.5 21.9 5320.27 5618.96 grr4mg7ms81

Module: DBMS\_SCHEDULER

INSERT INTO "ADDRESS\_RDONLY" ("ADDRESSID","ADDRESSTYPEID","CUSTOMERID","

ADDRESS1","ADDRESS2","ADDRESS3","CITY","ZIP","STATE","PHONECOUNTRYCODE","PHONENU

854,035,223 168 5,083,543.0 18.0 5713.50 7458.95 4at7cbx8hnz

SELECT "CUSTOMERID",CURRENT$."ISACTIVE",CURRENT$."FIRSTNAME",CURRENT$."LASTNAME",CU<

RRENT$."ORGANIZATION",CURRENT$."DATEREGISTERED",CURRENT$."CUSTOMERSTATUSID",CURR

ENT$."LASTMODIFIEDDATE",CURRENT$."SOURCE",CURRENT$."EMPLOYEEDEPT",CURRENT$.

对这些Top SQL,可以手工调优，也可以调用SQL Tuning Advisor。

参照以下文档：

Document 271196.1 Automatic SQL Tuning - SQL Profiles.

Document 262687.1 How to use the Sql Tuning Advisor.

Document 276103.1 PERFORMANCE TUNING USING ADVISORS AND MANAGEABILITY FEATURES: AWR, ASH, and ADDM and Sql Tuning Advisor.

注: 使用SQL Tuning Advisor需要额外的Oracle Tuning Pack License:

http://docs.oracle.com/cd/E11882\_01/license.112/e10594/options.htm#DBLIC170

分析：

-> Total Buffer Gets: 4,745,943,815

假设这是一个一个小时的AWR报告，4,745,943,815是一个很大的值；所以需要进一步分析这个SQL是否使用了最优的执行计划

Individual Buffer Gets

上面的例子里单个的SQL的buffer get非常多，最少的那个都是8亿5千万。这三个SQL指向了两个不同的引起过多buffers的原因：

单次执行buffer gets过多

SQL\_ID为'5t1y1nvmwp2'和'4at7cbx8hnz'的SQL语句总共被执行了168次，但是每次执行引起的buffer gets超过500万。这两个SQL应该是主要的需要调优的候选者。

执行次数过多

SQL\_ID 'grr4mg7ms81' 每次执行只是引起16次buffer gets,减少这条SQL每次执行的buffer get可能并不能显著减少总共的buffer gets。这条语句的问题是它执行的太频繁了，6500万次。

改变这条SQL的执行次数可能会更有意义。这个SQL看起来是在一个循环里面被调用，如果可以让它一次处理的数据更多也许可以减少它执行的次数。

注意：对于某些非常繁忙的系统来讲，以上的数字可能都是正常的。这时候我们需要把这些数字跟正常时段的数字作对比，如果没有什么太大差别，那么这些SQL并不是引起问题的元凶（虽然通过调优这些SQL我们仍然可以受益）

Other SQL Statistic Sections

就像之前提到的那样，AWR报告中有很多不同的部分用来分析各种不同的问题。如果特定的问题并没有出现，那么分析AWR报告的这些部分并不能有很大的帮助。

下面提到了一些可能的问题：

Waits for 'Cursor: mutex/pin'

如 果发现了一些像"Cursor: pin S wait on X" 或"Cursor: mutex X" 类的mutex等待，那么可能是由于parsing引起的问题。检查"SQL ordered by Parse Calls" 和"SQL ordered by Version Count"部分的Top SQL，这些SQL可能引起这类的问题。

以下文档可以帮助我们分析这类问题：

Document 1356828.1 FAQ: 'cursor: mutex ..' / 'cursor: pin ..' / 'library cache: mutex ..' Type Wait Events

Note:1349387.1 Troubleshooting 'cursor: pin S wait on X' waits.

Load Profile

根据Top 5等待事件的不同，"Load Profile"可以提供一些有用的背景资料或潜在问题的细节信息。

Load Profile

~~~~~~~~~~~~ Per Second Per Transaction

--------------- ---------------

Redo size: 4,585,414.80 3,165,883.14

Logical reads: 94,185.63 65,028.07

Block changes: 40,028.57 27,636.71

Physical reads: 2,206.12 1,523.16

Physical writes: 3,939.97 2,720.25

User calls: 50.08 34.58

Parses: 26.96 18.61

Hard parses: 1.49 1.03

Sorts: 18.36 12.68

Logons: 0.13 0.09

Executes: 4,925.89 3,400.96

Transactions: 1.45

% Blocks changed per Read: 42.50 Recursive Call %: 99.19

Rollback per transaction %: 59.69 Rows per Sort: 1922.64

在这个例子里，Top 5 Events部分显示问题可能跟SQL的执行有关，那么我们接下来检查load profile部分。

如果您检查AWR report是为了一般性的性能调优，那么可以看到有比较多的redo activity和比较高的physical writes. Physical writes比physical read要高，并且有42%的块被更改了.

此外，hard parse的次数要少于soft parse.

如果mutex等待事件比较严重，如"library cache: mutex X"，那么查看所有parse的比率会更有用。

当然，如果把Load Profile部分跟正常时候的AWR报告做比较会更有用，比如，比较redo size, users calls, 和 parsing这些性能指标。

Instance Efficiency

Instance Efficiency部分更适用于一般性的调优，而不是解决某个具体问题（除非等待事件直接指向这些指标）。

Instance Efficiency Percentages (Target 100%)

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Buffer Nowait %: 99.91 Redo NoWait %: 100.00

Buffer Hit %: 98.14 In-memory Sort %: 99.98

Library Hit %: 99.91 Soft Parse %: 94.48

Execute to Parse %: 99.45 Latch Hit %: 99.97

Parse CPU to Parse Elapsd %: 71.23 % Non-Parse CPU: 99.00

从我们的这个例子来看，最有用的信息是%Non-Parse CPU，它表明几乎所有的CPU都消耗在了Execution而不是Parse上，所以调优SQL会对性能有改善。

94.48％ 的soft parse比率显示hard parse的比例很小，这是可取的。Execute to Parse %很高，说明cursor被很好的重用了。我们总是期望这里的值都是接近100%,但是因为应用的不同，如果这个部分的参数的某些值很小，也是可以认为没 有问题的；如在数据仓库环境，hard parse因为使用了物化视图或histogram而变得很高。所以，重要的是，我们需要把这部分信息和正常时候的AWR报告做比较来判断是否有问题。

Latch Activity

在我们这个例子里，我们并没有看到很高的latch相关的等待，所以这部分的信息可以忽略。

但是如果latch相关的等待很严重，我们需要查看Latch Sleep Breakdown 部分sleeps很高的latch

Latch Sleep Breakdown

\* ordered by misses desc

Latch Name

----------------------------------------

Get Requests Misses Sleeps Spin Gets Sleep1 Sleep2 Sleep3

-------------- ----------- ----------- ---------- -------- -------- --------

cache buffers chains

2,881,936,948 3,070,271 41,336 3,031,456 0 0 0

row cache objects

941,375,571 1,215,395 852 1,214,606 0 0 0

object queue header operation

763,607,977 949,376 30,484 919,782 0 0 0

cache buffers lru chain

376,874,990 705,162 3,192 702,090 0 0 0

这 里top latch是cache buffers chains. Cache Buffers Chains latches是用来保护buffer caches中的buffers。在我们读取数据时，这个latch是正常需要获得的。Sleep的数字上升代表session在读取buffers时开 始等待这个latch。争用通常来自于不良的SQL要读取相同的buffers。

在我们这个例子里，虽然读取buffer的操作发生了 28亿次，但是只sleep了41,336次，可以认为是比较低的。Avg Slps/Miss（Sleeps/ Misses）也比较低。这表明当前Server有能力处理这样多的数据，所以没有发生Cache Buffers Chains latch的争用。

关于其他的latch free等待，请参照以下文档：

Note:413942.1 How to Identify Which Latch is Associated with a "latch free" wait

值得注意的wait events

CPU time events

CPU变为top wait event并不总是代表出现了问题。但是如果同时数据库性能比较慢，那么就需要进一步分析了。首先，检查AWR报告的“ SQL ordered by CPU Time ”部分，看是否某个特定的SQL使用了大量的CPU。

SQL ordered by CPU Time

-> Resources reported for PL/SQL code includes the resources used by all SQL

statements called by the code.

-> % Total is the CPU Time divided into the Total CPU Time times 100

-> Total CPU Time (s): 56,207

-> Captured SQL account for 114.6% of Total

CPU Elapsed CPU per % Total

Time (s) Time (s) Executions Exec (s) % Total DB Time SQL Id

---------- ---------- ------------ ----------- ------- ------- -------------

20,349 24,884 168 121.12 36.2 9.1 7bbhgqykv3cm9

Module: DBMS\_SCHEDULER

DECLARE job BINARY\_INTEGER := :job; next\_date TIMESTAMP WITH TIME ZONE := :myda

te; broken BOOLEAN := FALSE; job\_name VARCHAR2(30) := :job\_name; job\_subname

VARCHAR2(30) := :job\_subname; job\_owner VARCHAR2(30) := :job\_owner; job\_start

TIMESTAMP WITH TIME ZONE := :job\_start; job\_scheduled\_start TIMESTAMP WITH TIME

Analysis:

-> Total CPU Time (s): 56,207

它代表了15分钟的CPU time。但是这个数字是否有问题取决于整个报告的时间。

Top SQL使用的CPU是 20,349秒（大概5分钟）

整个CPU时间占DB Time的9.1%

执行了168次，这个执行次数跟之前提到的几个SQL是一样的，说明这些SQL可能都是被同一个JOB调用的。

其他潜在的CPU相关的问题：

检查是否有其他等待事件与高CPU 事件同时出现

如cursor: pin S问题可能引起高CPU使用：

Note:6904068.8 Bug 6904068 - High CPU usage when there are "cursor: pin S" waits

数据库以外的CPU使用率过高

如果一个数据库以外的进程使用了过多CPU，那么数据库进程能够获得的CPU就会减少，数据库性能就会受到影响。在这种情况下，运行OSWather或者其他的OS工具去发现是哪个进程使用了过多CPU

Note:433472.1 OS Watcher For Windows (OSWFW) User Guide

诊断CPU使用率

下面的文档进一步描述了如何进一步分析CPU问题:

Note:164768.1 Troubleshooting: High CPU Utilization

'Log file sync' waits

当 一个user session commit或rollback时，log writer进程会把redo从log buffer中写入redo logfile文件。AWR报告会帮助我们来确定是否存在这方面的问题，并且确认是否是由物理IO引起。如果”log file sync”事件比较严重，下面的文档详细描述了如何来处理:

Document 1376916.1 Troubleshooting: "Log File Sync" Waits

Note:34592.1WAITEVENT: "log file sync"

Buffer busy waits

当 一个session从buffer cache读取一个buffer时，如果这个buffer处于busy的状态（由于其它session正在向其中读取数据，或者是由于这个buffer被 其它的session以不兼容模式持有），那么这个session就会等待这个事件。参照下面文档来找出哪个block处于busy状态和为什么：

Document 155971.1 Resolving Intense and "Random" Buffer Busy Wait Performance Problems:Note:34405.1 WAITEVENT: "buffer busy waits"

诊断其他问题

关于其他性能问题，请参照文档:

Document 1377446.1 Troubleshooting Performance Issues

使用ADDM的报告

当分析性能问题时，除了AWR报告，我们还可以同时参照ADDM报告，对于潜在的性能问题，它同时提供了具体的解决方案建议。下面是从如下文档拿到的一个ADDM报告示例：

Note:250655.1How to use the Automatic Database Diagnostic Monitor:

Example Output:

DETAILED ADDM REPORT FOR TASK 'SCOTT\_ADDM' WITH ID 5

----------------------------------------------------

Analysis Period: 17-NOV-2003 from 09:50:21 to 10:35:47

Database ID/Instance: 494687018/1

Snapshot Range: from 1 to 3

Database Time: 4215 seconds

Average Database Load: 1.5 active sessions

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

FINDING 1: 65% impact (2734 seconds)

------------------------------------

PL/SQL execution consumed significant database time.

RECOMMENDATION 1: SQL Tuning, 65% benefit (2734 seconds)

ACTION: Tune the PL/SQL block with SQL\_ID fjxa1vp3yhtmr. Refer to

the "Tuning PL/SQL Applications" chapter of Oracle's "PL/SQL

User's Guide and Reference"

RELEVANT OBJECT: SQL statement with SQL\_ID fjxa1vp3yhtmr

BEGIN EMD\_NOTIFICATION.QUEUE\_READY(:1, :2, :3); END;

FINDING 2: 35% impact (1456 seconds)

------------------------------------

SQL statements consuming significant database time were found.

RECOMMENDATION 1: SQL Tuning, 35% benefit (1456 seconds)

ACTION: Run SQL Tuning Advisor on the SQL statement with SQL\_ID

gt9ahqgd5fmm2.

RELEVANT OBJECT: SQL statement with SQL\_ID gt9ahqgd5fmm2 and

PLAN\_HASH 547793521

UPDATE bigemp SET empno = ROWNUM

FINDING 3: 20% impact (836 seconds)

-----------------------------------

The throughput of the I/O subsystem was significantly lower than expected.

RECOMMENDATION 1: Host Configuration, 20% benefit (836 seconds)

ACTION: Consider increasing the throughput of the I/O subsystem.

Oracle's recommended solution is to stripe all data file using

the SAME methodology. You might also need to increase the

number of disks for better performance.

RECOMMENDATION 2: Host Configuration, 14% benefit (584 seconds)

ACTION: The performance of file

D:\ORACLE\ORADATA\V1010\UNDOTBS01.DBF was significantly worse

than other files. If striping all files using the SAME

methodology is not possible, consider striping this file over

multiple disks.

RELEVANT OBJECT: database file

"D:\ORACLE\ORADATA\V1010\UNDOTBS01.DBF"

SYMPTOMS THAT LED TO THE FINDING:

Wait class "User I/O" was consuming significant database time.

(34% impact [1450 seconds])

FINDING 4: 11% impact (447 seconds)

-----------------------------------

Undo I/O was a significant portion (33%) of the total database I/O.

NO RECOMMENDATIONS AVAILABLE

SYMPTOMS THAT LED TO THE FINDING:

The throughput of the I/O subsystem was significantly lower than

expected. (20% impact [836 seconds])

Wait class "User I/O" was consuming significant database time.

(34% impact [1450 seconds])

FINDING 5: 9.9% impact (416 seconds)

------------------------------------

Buffer cache writes due to small log files were consuming significant

database time.

RECOMMENDATION 1: DB Configuration, 9.9% benefit (416 seconds)

ACTION: Increase the size of the log files to 796 M to hold at

least 20 minutes of redo information.

ADDM报告相比AWR报告来说，它提供了可读性更好的建议；当然应该同时参照ADDM报告和AWR报告来得到更准确地诊断。

其他的AWR参考文章

当阅读AWR报告的其他部分时，可以参照下面的一些文档:

Document 786554.1 How to Read PGA Memory Advisory Section in AWR and Statspack Reports

Document 754639.1 How to Read Buffer Cache Advisory Section in AWR and Statspack Reports

Document 1301503.1 Troubleshooting: AWR Snapshot Collection issues

Document 1363422.1 Automatic Workload Repository (AWR) Reports - Start Point

Statspack

AWR报告取代了旧有的staspack及bstat/estat报告，下面的这些文档概述了如何阅读statspack报告:

http://www.oracle.com/technetwork/database/focus-areas/performance/statspack-opm4-134117.pdf

Additional information can be found in the following articles:

Document 94224.1 FAQ- Statspack Complete Reference

Document 394937.1 Statistics Package (STATSPACK) Guide

Document 149113.1 Installing and Configuring StatsPack Package

Document 149121.1 Gathering a StatsPack snapshot

Document 228913.1 Systemwide Tuning using STATSPACK Reports