Oracle AWR报告详细分析

<https://blog.csdn.net/demonson/article/details/79474133>

AWR 是 Oracle  10g 版本 推出的新特性， 全称叫Automatic Workload Repository-自动负载信息库  
AWR 是通过对比两次快照(snapshot)收集到的统计信息，来生成报表数据，生成的报表包括多个部分。  
  
**WORKLOAD REPOSITORY report for**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **DB Name** | **DB Id** | **Instance** | **Inst num** | **Release** | **RAC** | **Host** |
| ICCI | 1314098396 | ICCI1 | 1 | 10.2.0.3.0 | YES | HPGICCI1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Snap Id** | **Snap Time** | **Sessions** | **Cursors/Session** |
| Begin Snap: | 2678 | 25-Dec-08 14:04:50 | 24 | 1.5 |
| End Snap: | 2680 | 25-Dec-08 15:23:37 | 26 | 1.5 |
| Elapsed: |  | 78.79 (mins) |  |  |
| DB Time: |  | 11.05 (mins) |  |  |

DB Time不包括Oracle后台进程消耗的时间。如果DB Time远远小于Elapsed时间，说明数据库比较空闲。  
db time= cpu time + wait time（不包含空闲等待） （非后台进程）  
说白了就是db time就是记录的服务器花在数据库运算(非后台进程)和等待(非空闲等待)上的时间  
DB time = cpu time + all of nonidle wait event time  
  
在79分钟里（其间收集了3次快照数据），数据库耗时11分钟，RDA数据中显示系统有8个逻辑CPU（4个物理CPU），  
平均每个CPU耗时1.4分钟，CPU利用率只有大约2%（1.4/79）。说明系统压力非常小。

列出下面这两个来做解释：  
Report A:  
Snap Id Snap Time Sessions Curs/Sess  
--------- ------------------- -------- ---------  
Begin Snap: 4610 24-Jul-08 22:00:54 68 19.1  
End Snap: 4612 24-Jul-08 23:00:25 17 1.7  
Elapsed: 59.51 (mins)  
DB Time: 466.37 (mins)  
  
Report B:  
Snap Id Snap Time Sessions Curs/Sess  
--------- ------------------- -------- ---------  
Begin Snap: 3098 13-Nov-07 21:00:37 39 13.6  
End Snap: 3102 13-Nov-07 22:00:15 40 16.4  
Elapsed: 59.63 (mins)  
DB Time: 19.49 (mins)  
  
  
  
服务器是AIX的系统，4个双核cpu,共8个核:

/sbin> bindprocessor -q  
The available processors are: 0 1 2 3 4 5 6 7  
  
  
先说Report A,在snapshot间隔中，总共约60分钟，cpu就共有60\*8=480分钟，DB time为466.37分钟  
则：cpu花费了466.37分钟在处理Oralce非空闲等待和运算上(比方逻辑读)  
也就是说cpu有 466.37/480\*100% 花费在处理Oracle的操作上，这还不包括后台进程  
看Report B，总共约60分钟，cpu有 19.49/480\*100% 花费在处理Oracle的操作上  
很显然，Report B中服务器的平均负载很低。  
从awr report的Elapsed time和DB Time就能大概了解db的负载。  
  
  
可是对于批量系统，数据库的工作负载总是集中在一段时间内。如果快照周期不在这一段时间内，  
或者快照周期跨度太长而包含了大量的数据库空闲时间，所得出的分析结果是没有意义的.  
这也说明选择分析时间段很关键，要选择能够代表性能问题的时间段。

# Report Summary

## Cache Sizes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Begin** | **End** |  |  |
| Buffer Cache: | 3,344M | 3,344M | Std Block Size: | 8K |
| Shared Pool Size: | 704M | 704M | Log Buffer: | 14,352K |

显示SGA中每个区域的大小（在AMM改变它们之后），可用来与初始参数值比较。

shared pool主要包括library cache和dictionary cache。  
library cache用来存储最近解析（或编译）后SQL、PL/SQL和Java classes等。  
dictionary cache用来存储最近引用的数据字典。  
发生在library cache或dictionary cache的cache miss代价要比发生在buffer cache的代价高得多。  
因此shared pool的设置要确保最近使用的数据都能被cache。

## Load Profile

|  |  |  |
| --- | --- | --- |
|  | **Per Second** | **Per Transaction** |
| Redo size: | 918,805.72 | 775,912.72 |
| Logical reads: | 3,521.77 | 2,974.06 |
| Block changes: | 1,817.95 | 1,535.22 |
| Physical reads: | 68.26 | 57.64 |
| Physical writes: | 362.59 | 306.20 |
| User calls: | 326.69 | 275.88 |
| Parses: | 38.66 | 32.65 |
| Hard parses: | 0.03 | 0.03 |
| Sorts: | 0.61 | 0.51 |
| Logons: | 0.01 | 0.01 |
| Executes: | 354.34 | 299.23 |
| Transactions: | 1.18 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| % Blocks changed per Read: | 51.62 | Recursive Call %: | 51.72 |
| Rollback per transaction %: | 85.49 | Rows per Sort: | ######## |

显示数据库负载概况，将之与基线数据比较才具有更多的意义，如果每秒或每事务的负载变化不大，说明应用运行比较稳定。  
单个的报告数据只说明应用的负载情况，绝大多数据并没有一个所谓“正确”的值，然而  
Logons大于每秒1~2个、Hard parses大于每秒100、全部parses超过每秒300表明可能有争用问题。

Redo size：每秒产生的日志大小(单位字节)，可标志数据变更频率, 数据库任务的繁重与否。  
  
Logical reads：每秒/每事务逻辑读的块数.平决每秒产生的逻辑读的block数。Logical Reads= Consistent Gets + DB Block Gets

Block changes：每秒/每事务修改的块数

Physical reads：每秒/每事务物理读的块数

Physical writes：每秒/每事务物理写的块数

User calls：每秒/每事务用户call次数

Parses：SQL解析的次数.每秒解析次数，包括fast parse，soft parse和hard parse三种数量的综合。   
软解析每秒超过300次意味着你的"应用程序"效率不高，调整session\_cursor\_cache。  
在这里，fast parse指的是直接在PGA中命中的情况（设置了session\_cached\_cursors=n）；  
soft parse是指在shared pool中命中的情形；hard parse则是指都不命中的情况。

Hard parses：其中硬解析的次数，硬解析太多，说明SQL重用率不高。  
每秒产生的硬解析次数, 每秒超过100次，就可能说明你绑定使用的不好，也可能是共享池设置不合理。  
这时候可以启用参数cursor\_sharing=similar|force，该参数默认值为exact。但该参数设置为similar时，存在bug，可能导致执行计划的不优。

Sorts：每秒/每事务的排序次数

Logons：每秒/每事务登录的次数

Executes：每秒/每事务SQL执行次数

Transactions：每秒事务数.每秒产生的事务数，反映数据库任务繁重与否。

Blocks changed per Read：表示逻辑读用于修改数据块的比例.在每一次逻辑读中更改的块的百分比。

Recursive Call：递归调用占所有操作的比率.递归调用的百分比，如果有很多PL/SQL，那么这个值就会比较高。

Rollback per transaction：每事务的回滚率.看回滚率是不是很高，因为回滚很耗资源 ,如果回滚率过高,  
可能说明你的数据库经历了太多的无效操作 ,过多的回滚可能还会带来Undo Block的竞争   
该参数计算公式如下: Round(User rollbacks / (user commits + user rollbacks) ,4)\* 100% 。

Rows per Sort：每次排序的行数

注:

Oracle的硬解析和软解析

　　提到软解析(soft prase)和硬解析(hard prase)，就不能不说一下Oracle对sql的处理过程。  
当你发出一条sql语句交付Oracle，在执行和获取结果前，Oracle对此sql将进行几个步骤的处理过程：

　　1、语法检查(syntax check)

　　检查此sql的拼写是否语法。

　　2、语义检查(semantic check)

　　诸如检查sql语句中的访问对象是否存在及该用户是否具备相应的权限。

　　3、对sql语句进行解析(prase)

　　利用内部算法对sql进行解析，生成解析树(parse tree)及执行计划(execution plan)。

　　4、执行sql，返回结果(execute and return)

　　其中，软、硬解析就发生在第三个过程里。

　　Oracle利用内部的hash算法来取得该sql的hash值，然后在library cache里查找是否存在该hash值；

　　假设存在，则将此sql与cache中的进行比较；

　　假设“相同”，就将利用已有的解析树与执行计划，而省略了优化器的相关工作。这也就是软解析的过程。

　　诚然，如果上面的2个假设中任有一个不成立，那么优化器都将进行创建解析树、生成执行计划的动作。这个过程就叫硬解析。

创建解析树、生成执行计划对于sql的执行来说是开销昂贵的动作，所以，应当极力避免硬解析，尽量使用软解析。

## Instance Efficiency Percentages (Target 100%)

|  |  |  |  |
| --- | --- | --- | --- |
| Buffer Nowait %: | 100.00 | Redo NoWait %: | 100.00 |
| Buffer Hit %: | 98.72 | In-memory Sort %: | 99.86 |
| Library Hit %: | 99.97 | Soft Parse %: | 99.92 |
| Execute to Parse %: | 89.09 | Latch Hit %: | 99.99 |
| Parse CPU to Parse Elapsd %: | 7.99 | % Non-Parse CPU: | 99.95 |

本节包含了Oracle关键指标的内存命中率及其它数据库实例操作的效率。其中Buffer Hit Ratio 也称Cache Hit Ratio，  
Library Hit ratio也称Library Cache Hit ratio。  
同Load Profile一节相同，这一节也没有所谓“正确”的值，而只能根据应用的特点判断是否合适。  
在一个使用直接读执行大型并行查询的DSS环境，20%的Buffer Hit Ratio是可以接受的，而这个值对于一个OLTP系统是完全不能接受的。  
根据Oracle的经验，对于OLTP系统，Buffer Hit Ratio理想应该在90%以上。  
  
**Buffer Nowait**表示在内存获得数据的未等待比例。在缓冲区中获取Buffer的未等待比率  
Buffer Nowait的这个值一般需要大于99%。否则可能存在争用，可以在后面的等待事件中进一步确认。

**buffer hit**表示进程从内存中找到数据块的比率，监视这个值是否发生重大变化比这个值本身更重要。  
对于一般的OLTP系统，如果此值低于80%，应该给数据库分配更多的内存。  
数据块在数据缓冲区中的命中率，通常应在95%以上。否则，小于95%，需要调整重要的参数，小于90%可能是要加db\_cache\_size。  
一个高的命中率，不一定代表这个系统的性能是最优的，比如大量的非选择性的索引被频繁访问，就会造成命中率很高的假相（大量的db file sequential read）  
但是一个比较低的命中率，一般就会对这个系统的性能产生影响，需要调整。命中率的突变，往往是一个不好的信息。  
如果命中率突然增大，可以检查top buffer get SQL，查看导致大量逻辑读的语句和索引，  
如果命中率突然减小，可以检查top physical reads SQL，检查产生大量物理读的语句，主要是那些没有使用索引或者索引被删除的。

**Redo NoWait**表示在LOG缓冲区获得BUFFER的未等待比例。如果太低（可参考90%阀值），考虑增加LOG BUFFER。  
当redo buffer达到1M时，就需要写到redo log文件，所以一般当redo buffer设置超过1M，不太可能存在等待buffer空间分配的情况。  
当前，一般设置为2M的redo buffer，对于内存总量来说，应该不是一个太大的值。

**library hit**表示Oracle从Library Cache中检索到一个解析过的SQL或PL/SQL语句的比率，当应用程序调用SQL或存储过程时，  
Oracle检查Library Cache确定是否存在解析过的版本，如果存在，Oracle立即执行语句；如果不存在，Oracle解析此语句，并在Library Cache中为它分配共享SQL区。  
低的library hit ratio会导致过多的解析，增加CPU消耗，降低性能。  
如果library hit ratio低于90%，可能需要调大shared pool区。  
STATEMENT在共享区的命中率，通常应该保持在95%以上，否则需要要考虑：加大共享池；使用绑定变量；修改cursor\_sharing等参数。

**Latch Hit：**Latch是一种保护内存结构的锁，可以认为是SERVER进程获取访问内存数据结构的许可。  
要确保Latch Hit>99%，否则意味着Shared Pool latch争用，可能由于未共享的SQL，或者Library Cache太小，可使用绑定变更或调大Shared Pool解决。  
要确保>99%，否则存在严重的性能问题。当该值出现问题的时候，我们可以借助后面的等待时间和latch分析来查找解决问题。

**Parse CPU to Parse Elapsd：**解析实际运行时间/(解析实际运行时间+解析中等待资源时间)，越高越好。  
计算公式为：Parse CPU to Parse Elapsd %= 100\*(parse time cpu / parse time elapsed)。  
即：解析实际运行时间/(解析实际运行时间+解析中等待资源时间)。如果该比率为100%，意味着CPU等待时间为0，没有任何等待。

**Non-Parse CPU ：**SQL实际运行时间/(SQL实际运行时间+SQL解析时间)，太低表示解析消耗时间过多。  
计算公式为：% Non-Parse CPU =round(100\*1-PARSE\_CPU/TOT\_CPU),2)。如果这个值比较小，表示解析消耗的CPU时间过多。  
与PARSE\_CPU相比，如果TOT\_CPU很高，这个比值将接近100%，这是很好的，说明计算机执行的大部分工作是执行查询的工作，而不是分析查询的工作。

**Execute to Parse：**是语句执行与分析的比例，如果要SQL重用率高，则这个比例会很高。该值越高表示一次解析后被重复执行的次数越多。  
计算公式为：Execute to Parse =100 \* (1 - Parses/Executions)。  
本例中，差不多每execution 5次需要一次parse。所以如果系统Parses > Executions，就可能出现该比率小于0的情况。  
该值<0通常说明shared pool设置或者语句效率存在问题，造成反复解析，reparse可能较严重,或者是可能同snapshot有关，通常说明数据库性能存在问题。

**In-memory Sort：**在内存中排序的比率，如果过低说明有大量的排序在临时表空间中进行。  
考虑调大PGA(10g)。如果低于95%，可以通过适当调大初始化参数PGA\_AGGREGATE\_TARGET或者SORT\_AREA\_SIZE来解决，  
注意这两个参数设置作用的范围时不同的，SORT\_AREA\_SIZE是针对每个session设置的，PGA\_AGGREGATE\_TARGET则时针对所有的sesion的。

**Soft Parse：**软解析的百分比（softs/softs+hards），近似当作sql在共享区的命中率，太低则需要调整应用使用绑定变量。  
sql在共享区的命中率，小于<95%,需要考虑绑定，如果低于80%，那么就可以认为sql基本没有被重用。

## Shared Pool Statistics

|  |  |  |
| --- | --- | --- |
|  | **Begin** | **End** |
| Memory Usage %: | 47.19 | 47.50 |
| % SQL with executions>1: | 88.48 | 79.81 |
| % Memory for SQL w/exec>1: | 79.99 | 73.52 |

Memory Usage %：对于一个已经运行一段时间的数据库来说，共享池内存使用率，应该稳定在75%-90%间，  
如果太小，说明Shared Pool有浪费，而如果高于90，说明共享池中有争用，内存不足。  
这个数字应该长时间稳定在75%～90%。如果这个百分比太低，表明共享池设置过大，带来额外的管理上的负担，从而在某些条件下会导致性能的下降。  
如果这个百分率太高，会使共享池外部的组件老化，如果SQL语句被再次执行，这将使得SQL语句被硬解析。  
在一个大小合适的系统中，共享池的使用率将处于75%到略低于90%的范围内.

SQL with executions>1：执行次数大于1的sql比率，如果此值太小，说明需要在应用中更多使用绑定变量，避免过多SQL解析。  
在一个趋向于循环运行的系统中，必须认真考虑这个数字。在这个循环系统中，在一天中相对于另一部分时间的部分时间里执行了一组不同的SQL语句。  
在共享池中，在观察期间将有一组未被执行过的SQL语句，这仅仅是因为要执行它们的语句在观察期间没有运行。只有系统连续运行相同的SQL语句组，这个数字才会接近100%。

Memory for SQL w/exec>1：执行次数大于1的SQL消耗内存的占比。  
这是与不频繁使用的SQL语句相比，频繁使用的SQL语句消耗内存多少的一个度量。  
这个数字将在总体上与% SQL with executions>1非常接近，除非有某些查询任务消耗的内存没有规律。  
在稳定状态下，总体上会看见随着时间的推移大约有75%～85%的共享池被使用。如果Statspack报表的时间窗口足够大到覆盖所有的周期，  
执行次数大于一次的SQL语句的百分率应该接近于100%。这是一个受观察之间持续时间影响的统计数字。可以期望它随观察之间的时间长度增大而增大。

小结：通过ORACLE的实例有效性统计数据，我们可以获得大概的一个整体印象，然而我们并不能由此来确定数据运行的性能。当前性能问题的确定，  
我们主要还是依靠下面的等待事件来确认。我们可以这样理解两部分的内容，hit统计帮助我们发现和预测一些系统将要产生的性能问题，由此我们  
可以做到未雨绸缪。而wait事件，就是表明当前数据库已经出现了性能问题需要解决，所以是亡羊补牢的性质。

## Top 5 Timed Events

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Event** | **Waits** | **Time(s)** | **Avg Wait(ms)** | **% Total Call Time** | **Wait Class** |
| CPU time |  | 515 |  | 77.6 |  |
| SQL\*Net more data from client | 27,319 | 64 | 2 | 9.7 | Network |
| log file parallel write | 5,497 | 47 | 9 | 7.1 | System I/O |
| db file sequential read | 7,900 | 35 | 4 | 5.3 | User I/O |
| db file parallel write | 4,806 | 34 | 7 | 5.1 | System I/O |

这是报告概要的最后一节，显示了系统中最严重的5个等待，按所占等待时间的比例倒序列示。当我们调优时，总希望观察到最显著的效果，  
因此应当从这里入手确定我们下一步做什么。  
例如**如果‘buffer busy wait’是较严重的等待事件，我们应当继续研究报告中Buffer Wait和File/Tablespace IO区的内容，  
识别哪些文件导致了问题。如果最严重的等待事件是I/O事件，我们应当研究按物理读排序的SQL语句区以识别哪些语句在  
执行大量I/O，并研究Tablespace和I/O区观察较慢响应时间的文件。如果有较高的LATCH等待，就需要察看详细的LATCH  
统计识别哪些LATCH产生的问题。**

一个性能良好的系统，cpu time应该在top 5的前面，否则说明你的系统大部分时间都用在等待上。

在这里，log file parallel write是相对比较多的等待，占用了7%的CPU时间。

通常，在没有问题的数据库中，CPU time总是列在第一个。

更多的等待事件，参见本报告 的Wait Events一节。

# RAC Statistics

|  |  |  |
| --- | --- | --- |
|  | **Begin** | **End** |
| Number of Instances: | 2 | 2 |

## Global Cache Load Profile

|  |  |  |
| --- | --- | --- |
|  | **Per Second** | **Per Transaction** |
| Global Cache blocks received: | 4.16 | 3.51 |
| Global Cache blocks served: | 5.97 | 5.04 |
| GCS/GES messages received: | 408.47 | 344.95 |
| GCS/GES messages sent: | 258.03 | 217.90 |
| DBWR Fusion writes: | 0.05 | 0.05 |
| Estd Interconnect traffic (KB) | 211.16 |  |

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

|  |  |
| --- | --- |
| Buffer access - local cache %: | 98.60 |
| Buffer access - remote cache %: | 0.12 |
| Buffer access - disk %: | 1.28 |

## Global Cache and Enqueue Services - Workload Characteristics

|  |  |
| --- | --- |
| Avg global enqueue get time (ms): | 0.1 |
| Avg global cache cr block receive time (ms): | 1.1 |
| Avg global cache current block receive time (ms): | 0.8 |
| 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 %: | 3.5 |
| Avg global cache cr block flush time (ms): | 3.9 |
| Avg global cache current block pin time (ms): | 0.0 |
| Avg global cache current block send time (ms): | 0.0 |
| Global cache log flushes for current blocks served %: | 0.4 |
| Avg global cache current block flush time (ms): | 3.0 |

## Global Cache and Enqueue Services - Messaging Statistics

|  |  |
| --- | --- |
| Avg message sent queue time (ms): | 0.0 |
| Avg message sent queue time on ksxp (ms): | 0.3 |
| Avg message received queue time (ms): | 0.5 |
| Avg GCS message process time (ms): | 0.0 |
| Avg GES message process time (ms): | 0.0 |
| % of direct sent messages: | 14.40 |
| % of indirect sent messages: | 77.04 |
| % of flow controlled messages: | 8.56 |

# Main Report

* [**Wait Events Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#21)
* [**SQL Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#22)
* [**Instance Activity Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#23)
* [**IO Stats**](http://blog.itpub.net/26736162/viewspace-1869086/#24)
* [**Buffer Pool Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#1900)
* [**Advisory Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#26)
* [**Wait Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#27)
* [**Undo Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#28)
* [**Latch Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#29)
* [**Segment Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#30)
* [**Dictionary Cache Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#31)
* [**Library Cache Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#32)
* [**Memory Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#33)
* [**Streams Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#34)
* [**Resource Limit Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#6700)
* [**init.ora Parameters**](http://blog.itpub.net/26736162/viewspace-1869086/#6800)

## Wait Events Statistics

* [**Time Model Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#1500)
* [**Wait Class**](http://blog.itpub.net/26736162/viewspace-1869086/#200)
* [**Wait Events**](http://blog.itpub.net/26736162/viewspace-1869086/#100)
* [**Background Wait Events**](http://blog.itpub.net/26736162/viewspace-1869086/#300)
* [**Operating System Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#1600)
* [**Service Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#5800)
* [**Service Wait Class Stats**](http://blog.itpub.net/26736162/viewspace-1869086/#5820)

[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

/\* oracle等待事件是衡量oracle运行状况的重要依据及指示，等待事件分为两类：  
空闲等待事件和非空闲等待事件, TIMED\_STATISTICS = TRUE 那么等待事件按等待的时间排序，= FALSE那么事件按等待的数量排序。  
运行statspack期间必须session上设置TIMED\_STATISTICS = TRUE，否则统计的数据将失真。  
空闲等待事件是oracle正等待某种工作，在诊断和优化数据库时候，不用过多注意这部分事件，  
非空闲等待事件专门针对oracle的活动，指数据库任务或应用程序运行过程中发生的等待，  
**这些等待事件是我们在调整数据库应该关注的**。

    对于常见的等待事件，说明如下：

1)    **db file scattered read 文件分散读取**该事件通常与全表扫描或者fast full index scan有关。因为全表扫描是被放入内存中进行的进行的，通常情况下基于性能的考虑，有时候也可能是分配不到足够长的连续内存空间，所以会将数据块分散(scattered)读入Buffer Cache中。该等待过大可能是缺少索引或者没有合适的索引(可以调整optimizer\_index\_cost\_adj) 。这种情况也可能是正常的，因为执行全表扫描可能比索引扫描效率更高。当系统存在这些等待时，需要通过检查来确定全表扫描是否必需的来调整。因为全表扫描被置于LRU(Least Recently Used，最近最少适用)列表的冷端(cold end)，对于频繁访问的较小的数据表，可以选择把他们Cache 到内存中，以避免反复读取。当这个等待事件比较显著时，可以结合v$session\_longops 动态性能视图来进行诊断，该视图中记录了长时间(运行时间超过6 秒的)运行的事物，可能很多是全表扫描操作(不管怎样，这部分信息都是值得我们注意的)。  
关于参数OPTIMIZER\_INDEX\_COST\_ADJ＝n：该参数是一个百分比值，缺省值为100，可以理解为FULL SCAN COST/INDEX SCAN COST。当n%\* INDEX SCAN COST<FULL SCAN COST时，oracle会选择使用索引。在具体设置的时候，我们可以根据具体的语句来调整该值。如果我们希望某个statement使用索引，而实际它确走全表扫描，可以对比这两种情况的执行计划不同的COST，从而设置一个更合适的值。

2)    **db file sequential read 文件顺序读取**整代码,特别是表连接：该事件说明在单个数据块上大量等待，该值过高通常是由于表间连接顺序很糟糕（没有正确选择驱动行源），或者使用了非选择性索引。通过将这种等待与statspack报表中已知其它问题联系起来(如效率不高的sql),通过检查确保索引扫描是必须的，并确保多表连接的连接顺序来调整。

3)    **buffer busy wait 缓冲区忙**增大DB\_CACHE\_SIZE,加速检查点,调整代码：

当进程需要存取SGA中的buffer的时候，它会依次执行如下步骤的操作：

当缓冲区以一种非共享方式或者如正在被读入到缓冲时,就会出现该等待。该值不应该大于1%。当出 现等待问题时，可以检查缓冲等待统计部分(或V$WAITSTAT)，确定该等待发生在什么位置：

a)    如果等待是否位于段头(Segment Header)。这种情况表明段中的空闲列表（freelist）的块比较少。可以考虑增加空闲列表(freelist，对于Oracle8i DMT)或者增加freelist groups(在很多时候这个调整是立竿见影的(alter table tablename strorage(freelists 2))，在8.1.6之前，这个freelists参数不能动态修改;在8.1.6及以后版本，动态修改feelists需要设置COMPATIBLE至少为8.1.6)。也可以增加PCTUSED与PCTFREE之间距离（PCTUSED-to-pctfree gap），其实就是说降低PCTUSED的值，尽快使块返回freelist列表被重用。如果支持自动段空间管理（ASSM），也可以使用ASSM模式，这是在ORALCE 920以后的版本中新增的特性。

b)    如果这一等待位于undo header，可以通过增加回滚段(rollback segment)来解决缓冲区的问题。

c)    如果等待位于undo block上，我们需要增加提交的频率，使block可以尽快被重用；使用更大的回滚段；降低一致读所选择的表中数据的密度；增大DB\_CACHE\_SIZE。

d)    如果等待处于data block，表明出现了hot block，可以考虑如下方法解决： ①将频繁并发访问的表或数据移到另一数据块或者进行更大范围的分布(可以增大pctfree值 ，扩大数据分布，减少竞争)，以避开这个"热点"数据块。②也可以减小数据块的大小，从而减少一个数据块中的数据行数，降低数据块的热度，减小竞争；③检查对这些热块操作的SQL语句，优化语句。④增加hot block上的initrans值。但注意不要把initrans值设置的过于高了，通常设置为5就足够了。因为增加事务意味着要增加ITL事务槽，而每个ITL事务槽将占用数据块中24个字节长度。默认情况下，每个数据块或者索引块中是ITL槽是2个，在增加initrans的时候，可以考虑增大数据块所在的表的PCTFREE值，这样Oracle会利用PCTFREE部分的空间增加ITL slot数量，最大达到maxtrans指定。

e)    如果等待处于index block，应该考虑重建索引、分割索引或使用反向键索引。为了防止与数据块相关的缓冲忙等待，也可以使用较小的块，在这种情况下，单个块中的记录就较少，所以这个块就不是那么"繁忙"。或者可以设置更大的PCTFREE,使数据扩大物理分布，减少记录间的热点竞争。在执行DML (insert/update/ delete)时，Oracle向数据块中写入信息，对于多事务并发访问的数据表，关于ITL的竞争和等待可能出现，为了减少这个等待，可以增加initrans，使用多个ITL槽。在Oracle9i 中，可以使用ASSM这个新特性Oracle 使用位图来管理空间使用，减小争用。

当进程需要存取SGA中的buffer的时候，它会依次执行如下步骤的操作：  
1.获得cache buffers chains latch，遍历那条buffer chain直到找到需要的buffer header  
2.根据需要进行的操作类型(读或写)，它需要在buffer header上获得一个共享或独占模式的buffer pin或者buffer lock  
3.若进程获得buffer header pin，它会释放获得的cache buffers chains latch，然后执行对buffer block的操作  
4.若进程无法获得buffer header pin，它就会在buffer busy waits事件上等待  
  
进程之所以无法获得buffer header pin，是因为为了保证数据的一致性，同一时刻一个block只能被一个进程pin住进行存取，  
因此当一个进程需要存取buffer cache中一个被其他进程使用的block的时候，这个进程就会产生对该block的buffer busy waits事件。  
  
截至**Oracle** 9i，buffer busy waits事件的p1,p2,p3三个参数分别是file#,block#和id，分别表示等待的buffer block所在的文件编号，  
块编号和具体的等待原因编号，到了Oracle **10g**，前两个参数没变，第3个参数变成了块类型编号，这一点可以通过查询v$event\_name视图来进行验证：

Oracle 9i  
**SQL**> select parameter1,parameter2,parameter3 from v$event\_name where name='buffer busy waits'  
PARAMETER1                  PARAMETER2                 PARAMETER3  
------------------------ ------------------------ ------------------------  
file#                             block#                          id  
Oracle 10g  
PARAMETER1                  PARAMETER2                 PARAMETER3  
------------------------ ------------------------ ------------------------  
file#                             block#                          class#

在诊断buffer busy waits事件的过程中，获取如下信息会很有用：  
1.获取产生buffer busy waits事件的等待原因编号，这可以通过查询该事件的p3参数值获得  
2.获取产生此事件的SQL语句，可以通过如下的查询获得：  
select sql\_text from v$sql t1,v$session t2,v$session\_wait t3  
where t1.address=t2.sql\_address and t1.hash\_value=t2.sql\_hash\_value  
and t2.sid=t3.sid and t3.event='buffer busy waits';  
3.获取等待的块的类型以及所在的segment，可以通过如下查询获得：

select 'Segment Header' class,a.segment\_type,a.segment\_name,a.partition\_name from dba\_segments a,v$session\_wait b  
where a.header\_file=b.p1 and a.header\_block=b.p2 and b.event='buffer busy waits'  
union  
select 'Freelist Groups' class,a.segment\_type,a.segment\_name,a.partition\_name from dba\_segments a,v$session\_wait b  
where a.header\_file=b.p1 and b.p2 between a.header\_block+1 and (a.header\_block+a.freelist\_groups) and a.freelist\_groups>1 and b.event='buffer busy waits'  
union  
select a.segment\_type||' block' class,a.segment\_type,a.segment\_name,a.partition\_name from dba\_extents a,v$session\_wait b  
where a.file\_id=b.p1 and b.p2 between a.block\_id and a.block\_id+a.blocks-1 and b.event='buffer busy waits' and not exists(select 1 from dba\_segments where   
header\_file=b.p1 and header\_block= b.p2);

查询的第一部分：如果等待的块类型是segment header，那么可以直接拿buffer busy waits事件的p1和p2参数去dba\_segments视图中匹配header\_file和header\_block字段即可找到等待的segment名称和segment类型，进行相应调整  
查询的第二部分：如果等待的块类型是freelist groups，也可以在dba\_segments视图中找出对应的segment名称和segment类型，注意这里的参数p2表示的freelist groups的位置是在segment的header\_block+1到header\_block+freelist groups组数之间，并且freelist groups组数大于1  
查询的第三部分：如果等待的块类型是普通的数据块，那么可以用p1、p2参数和dba\_extents进行联合查询得到block所在的segment名称和segment类型  
  
对于不同的等待块类型，我们采取不同的处理办法：  
1.data segment header：  
进程经常性的访问**data** segment header通常有两个原因：获取或修改process freelists信息、扩展高水位标记，针对第一种情况，进程频繁访问process freelists信息导致freelist争用，我们可以增大相应的segment对象的存储参数freelist或者freelist groups；若由于数据块频繁进出freelist而导致进程经常要修改freelist，则可以将pctfree值和pctused值设置较大的差距，从而避免数据块频繁进出freelist；对于第二种情况，由于该segment空间消耗很快，而设置的next extent过小，导致频繁扩展高水位标记，解决的办法是增大segment对象的存储参数next extent或者直接在创建表空间的时候设置extent size uniform  
2.data block：  
某一或某些数据块被多个进程同时读写，成为热点块，可以通过如下这些办法来解决这个问题：  
(1)降低程序的并发度，如果程序中使用了parallel查询，降低parallel degree，以免多个parallel slave同时访问同样的数据对象而形成等待降低性能  
(2)调整应用程序使之能读取较少的数据块就能获取所需的数据，减少buffer gets和physical reads  
(3)减少同一个block中的记录数，使记录分布于更多的数据块中，这可以通过若干途径实现：可以调整segment对象的pctfree值，可以将segment重建到block size较小的表空间中，还可以用alter table minimize records\_per\_block语句减少每块中的记录数  
(4)若热点块对象是类似自增id字段的索引，则可以将索引转换为反转索引，打散数据分布，分散热点块  
3.undo segment header：  
undo segment header争用是因为系统中undo segment不够，需要增加足够的undo segment，根据undo segment的**管理**方法，若是手工管理模式，需要修改rollback\_segments初始化参数来增加rollback segment，若是自动管理模式，可以减小transactions\_per\_rollback\_segment初始化参数的值来使oracle自动增多rollback segment的数量  
4.undo block：  
undo block争用是由于应用程序中存在对数据的读和写同时进行，读进程需要到undo segment中去获得一致性数据，解决办法是错开应用程序修改数据和大量查询数据的时间  
  
小结：buffer busy waits事件是oracle**等待事件**中比较复杂的一个，其形成原因很多，需要根据p3参数对照Oracle提供的原因代码表进行相应的诊断，10g以后则需要根据等待的block类型结合引起等待时间的具体SQL进行分析，采取相应的调整措施

4)    **latch free**：当闩锁丢失率高于0.5%时，需要调整这个问题。详细的我们在后面的Latch Activity for DB部分说明。

latch是一种低级排队机制，用于保护SGA中共享内存结构。latch就像是一种快速地被获取和释放的内存锁。用于防止共享内存结构被多个用户同时访问。如果latch不可用，就会记录latch释放失败(latch free miss )。有两种与闩有关的类型:

　　■ 立刻。

　　■ 可以等待。

　　假如一个进程试图在立刻模式下获得闩，而该闩已经被另外一个进程所持有，如果该闩不能立可用的话，那么该进程就不会为获得该闩而等待。它将继续执行另一个操作。

　　大多数latch问题都与以下操作相关:

　　没有很好的是用绑定变量(library cache latch)、重作生成问题(redo allocation latch)、缓冲存储竞争问题(cache buffers LRU chain)，以及buffer cache中的存在"热点"块(cache buffers chain)。

　　通常我们说，如果想设计一个失败的系统，不考虑绑定变量，这一个条件就够了，对于异构性强的系统，不使用绑定变量的后果是极其严重的。

　　另外也有一些latch等待与bug有关，应当关注Metalink相关bug的公布及补丁的发布。当latch miss ratios大于0.5%时，就应当研究这一问题。

　　Oracle的latch机制是竞争，其处理类似于网络里的CSMA/CD，所有用户进程争夺latch， 对于愿意等待类型(willing-to-wait)的latch,如果一个进程在第一次尝试中没有获得latch,那么它会等待并且再尝试一次,如果经过\_spin\_count次争夺不能获得latch, 然后该进程转入睡眠状态，持续一段指定长度的时间，然后再次醒来，按顺序重复以前的步骤.在8i/9i中默认值是\_spin\_count=2000。

　　如果SQL语句不能调整，在8.1.6版本以上，Oracle提供了一个新的初始化参数: CURSOR\_SHARING可以通过设置CURSOR\_SHARING = force 在服务器端强制绑定变量。设置该参数可能会带来一定的副作用，对于Java的程序，有相关的bug，具体应用应该关注Metalink的bug公告。

\*\*\*Latch 问题及可能解决办法   
------------------------------   
\* Library Cache and Shared Pool (未绑定变量---绑定变量,调整shared\_pool\_size)   
每当执行SQL或PL/SQL存储过程,包,函数和触发器时,这个Latch即被用到.Parse操作中此Latch也会被频繁使用.   
\* Redo Copy (增大\_LOG\_SIMULTANEOUS\_COPIES参数)   
重做拷贝Latch用来从PGA向重做日志缓冲区拷贝重做记录.   
\* Redo Allocation (最小化REDO生成,避免不必要提交)   
此Latch用来分配重做日志缓冲区中的空间,可以用NOLOGGING来减缓竞争.   
\* Row Cache Objects (增大共享池)   
数据字典竞争.过度parsing.   
\* Cache Buffers Chains (\_DB\_BLOCK\_HASH\_BUCKETS应增大或设为质数)   
"过热"数据块造成了内存缓冲链Latch竞争.   
\* Cache Buffers Lru Chain (调整SQL，设置DB\_BLOCK\_LRU\_LATCHES,或使用多个缓冲区池)   
扫描全部内存缓冲区块的LRU(最近最少使用)链时要用到内存缓冲区LRU链Latch.太小内存缓冲区、过大的内存缓冲区吞吐量、过多的内存中进行的排序操作、DBWR速度跟不上工作负载等会引起此Latch竞争。

5)    Enqueue 队列是一种锁，保护一些共享资源，防止并发的DML操作。队列采用FIFO策略，注意latch并不是采用的FIFO机制。比较常见的有3种类型的队列：ST队列，HW队列，TX4队列。  
ST Enqueue的等待主要是在字典管理的表空间中进行空间管理和分配时产生的。解决方法：1）将字典管理的表空间改为本地管理模式 2）预先分配分区或者将有问题的字典管理的表空间的next extent设置大一些。  
HW Enqueue是用于segment的HWM的。当出现这种等待的时候，可以通过手工分配extents来解决。  
TX4 Enqueue等待是最常见的等待情况。通常有3种情况会造成这种类型的等待：1）唯一索引中的重复索引。解决方法：commit或者rollback以释放队列。 2）对同一个位图索引段（bitmap index fragment）有多个update，因为一个bitmap index fragment可能包含了多个rowid，所以当多个用户更新时，可能一个用户会锁定该段，从而造成等待。解决方法同上。3）有多个用户同时对一个数据块作update，当然这些DML操作可能是针对这个数据块的不同的行，如果此时没有空闲的ITL槽，就会产生一个block-level锁。解决方法：增大表的initrans值使创建更多的ITL槽；或者增大表的pctfree值，这样oracle可以根据需要在pctfree的空间创建更多的ITL槽；使用smaller block size，这样每个块中包含行就比较少，可以减小冲突发生的机会。

**AWR报告分析--等待事件-队列.doc**

6)    **Free Buffer 释放缓冲区**：这个等待事件表明系统正在等待内存中的可用空间，这说明当前Buffer 中已经没有Free 的内存空间。如果应用设计良好，SQL 书写规范，充分绑定变量，那这种等待可能说明Buffer Cache 设置的偏小，你可能需要增大DB\_CACHE\_SIZE。该等待也可能说明DBWR的写出速度不够，或者磁盘存在严重的竞争，可以需要考虑增加检查点、使用更多的DBWR 进程，或者增加物理磁盘的数量,分散负载，平衡IO。

7)    Log file single write：该事件仅与写日志文件头块相关，通常发生在增加新的组成员和增进序列号时。头块写单个进行，因为头块的部分信息是文件号，每个文件不同。更新日志文件头这个操作在后台完成，一般很少出现等待，无需太多关注。

8)    log file parallel write：从log buffer 写redo 记录到redo log 文件，主要指常规写操作(相对于log file sync)。如果你的Log group 存在多个组成员，当flush log buffer 时，写操作是并行的，这时候此等待事件可能出现。尽管这个写操作并行处理，直到所有I/O 操作完成该写操作才会完成(如果你的磁盘支持异步IO或者使用IO SLAVE，那么即使只有一个redo log file member,也有可能出现此等待)。这个参数和log file sync 时间相比较可以用来衡量log file 的写入成本。通常称为同步成本率。改善这个等待的方法是将redo logs放到I/O快的盘中，尽量不使用raid5，确保表空间不是处在热备模式下，确保redo log和data的数据文件位于不同的磁盘中。

9)    log file sync：当一个用户提交或回滚数据时，LGWR将会话的redo记录从日志缓冲区填充到日志文件中，用户的进程必须等待这个填充工作完成。在每次提交时都出现，如果这个等待事件影响到数据库性能，那么就需要修改应用程序的提交频率, 为减少这个等待事件，须一次提交更多记录，或者将重做日志REDO LOG 文件访在不同的物理磁盘上，提高I/O的性能。

当一个用户提交或回滚数据时，LGWR 将会话期的重做由日志缓冲器写入到重做日志中。日志文件同步过程必须等待这一过程成功完成。为了减少这种等待事件，可以尝试一次提交更多的记录(频繁的提交会带来更多的系统开销)。将重做日志置于较快的磁盘上，或者交替使用不同物理磁盘上的重做日志，以降低归档对LGWR的影响。

　　对于软RAID，一般来说不要使用RAID 5，RAID5 对于频繁写入得系统会带来较大的性能损失，可以考虑使用文件系统直接输入/输出，或者使用裸设备(raw device)，这样可以获得写入的性能提高。

10)  log buffer space：日志缓冲区写的速度快于LGWR写REDOFILE的速度,可以增大日志文件大小，增加日志缓冲区的大小，或者使用更快的磁盘来写数据。

当你将日志缓冲(log buffer)产生重做日志的速度比LGWR 的写出速度快，或者是当日志切换(log switch)太慢时，就会发生这种等待。这个等待出现时，通常表明redo log buffer 过小，为解决这个问题，可以考虑增大日志文件的大小，或者增加日志缓冲器的大小。

　　另外一个可能的原因是磁盘I/O 存在瓶颈，可以考虑使用写入速度更快的磁盘。在允许的条件下设置可以考虑使用裸设备来存放日志文件，提高写入效率。在一般的系统中，最低的标准是，不要把日志文件和数据文件存放在一起，因为通常日志文件只写不读，分离存放可以获得性能提升。

11)  **logfile switch**：通常是因为归档速度不够快。表示所有的提交(commit)的请求都需要等待"日志文件切换"的完成。Log file Switch 主要包含两个子事件:  
log file switch (archiving needed) 这个等待事件出现时通常是因为日志组循环写满以后，第一个日志归档尚未完成，出现该等待。出现该等待，可能表示io 存在问题。解决办法:①可以考虑增大日志文件和增加日志组；②移动归档文件到快速磁盘；③调整log\_archive\_max\_processes。  
log file switch (checkpoint incomplete) 当日志组都写完以后，LGWR 试图写第一个log file，如果这时数据库没有完成写出记录在第一个log file 中的dirty 块时(例如第一个检查点未完成)，该等待事件出现。该等待事件通常表示你的DBWR 写出速度太慢或者IO 存在问题。为解决该问题，你可能需要考虑增加额外的DBWR 或者增加你的日志组或日志文件大小，或者也可以考虑增加checkpoint的频率。

12)  DB File Parallel Write：文件被DBWR并行写时发生。解决办法：改善IO性能。

处理此事件时，需要注意

1）db file parallel write事件只属于DBWR进程。

2）缓慢的DBWR可能影响前台进程。

3）大量的db file parallel write等待时间很可能是I/O问题引起的。（在确认os支持异步io的前提下，你可以在系统中检查disk\_asynch\_io参数，保证为TRUE。可以通过设置db\_writer\_processes来提高DBWR进程数量，当然前提是不要超过cpu的数量。）  
     DBWR进程执行经过SGA的所有**数据库**写入，当开始写入时，DBWR进程编译一组脏块（dirty block），并且将系统写入调用发布到操作系统。DBWR进程查找在各个时间内写入的块，包括每隔3秒的一次查找，当前台进程提交以清除缓冲区中的内容时：在检查点处查找，当满足\_DB\_LARGE\_DIRTY\_QUEUE、\_DB\_BLOCK\_MAX\_DIRTY\_TARGET和FAST\_START\_MTTR\_TARGET阀值时，等等。  
    虽然用户会话从来没有经历过db file parallel write等待事件，但这并不意味着它们不会受到这种事件的影响。缓慢的DBWR写入性能可以造成前台会话在write complete waits或free buffer waits事件上等待。DBWR写入性能可能受到如下方面的影响：I/O操作的类型（同步或异步）、存储设备（裸设备或成熟的文件系统）、数据库布局和I/O子系统配置。需要查看的关键数据库统计是当db file parallel write、free buffer waits和write complete waits等待事件互相关联时，系统范围内的TIME\_WAITED和AVERAGE\_WAIT。  
    如果db file parallel write平均等待时间大于10cs（或者100ms），则通常表明缓慢的I/O吞吐量。可以通过很多方法来改善平均等待时间。主要的方法是使用正确类型的I/O操作。如果数据文件位于裸设备（raw device）上，并且平台支持异步I/O，就应该使用异步写入。但是，如果数据库位于文件系统上，则应该使用同步写入和直接I/O（这是操作系统直接I/O）。除了确保正在使用正确类型的I/O操作，还应该检查你的数据库布局并使用常见的命令监控来自操作系统的I/O吞吐量。例如sar -d或iostat -dxnC。  
    当db file parallel write平均等待时间高并且系统繁忙时，用户会话可能开始在free buffer waits事件上等待。这是因为DBWR进程不能满足释放缓冲区的需求。如果free buffer waits事件的TIME\_WAITED高，则应该在高速缓存中增加缓冲区数量之前说明DBWR I/O吞吐量的问题。  
    高db file parallel write平均等待时间的另一个反响是在write complete waits等待事件上的高TIME\_WAITED。前台进程不允许修改正在传输到磁盘的块。换句话说，也就是位于DBWR批量写入中的块。前台的会话在write complete waits等待事件上等待。因此，write complete waits事件的出现，一定标志着缓慢的DBWR进程，可以通过改进DBWR I/O吞吐量修正这种延迟。

13)  DB File Single Write：当文件头或别的单独块被写入时发生，这一等待直到所有的I/O调用完成。解决办法：改善IO性能。

14)  DB FILE Scattered Read：当扫描整个段来根据初始化参数db\_file\_multiblock\_read\_count读取多个块时发生，因为数据可能分散在不同的部分，这与分条或分段）相关，因此通常需要多个分散的读来读取所有的数据。等待时间是完成所有I/O调用的时间。解决办法：改善IO性能。

这种情况通常显示与全表扫描相关的等待。  
当数据库进行全表扫时，基于性能的考虑，数据会分散(scattered)读入Buffer Cache。如果这个等待事件比较显著，可能说明对于某些全表扫描的表，没有创建索引或者没有创建合适的索引，我们可能需要检查这些数据表已确定是否进行了正确的设置。

然而这个等待事件不一定意味着性能低下，在某些条件下Oracle会主动使用全表扫描来替换索引扫描以提高性能，这和访问的数据量有关，在CBO下Oracle会进行更为智能的选择，在RBO下Oracle更倾向于使用索引。

因为全表扫描被置于LRU（Least Recently Used，最近最少适用）列表的冷端（cold end），对于频繁访问的较小的数据表，可以选择把他们Cache到内存中，以避免反复读取。

当这个等待事件比较显著时，可以结合v$session\_longops动态性能视图来进行诊断，该视图中记录了长时间（运行时间超过6秒的）运行的事物，可能很多是全表扫描操作（不管怎样，这部分信息都是值得我们注意的）。

15)  DB FILE Sequential Read：当前台进程对数据文件进行常规读时发生，包括索引查找和别的非整段扫描以及数据文件块丢弃等待。等待时间是完成所有I/O调用的时间。解决办法：改善IO性能。

如果这个等待事件比较显著，可能表示在多表连接中，表的连接顺序存在问题，没有正确地使用驱动表；或者可能索引的使用存在问题，并非索引总是最好的选择。在大多数情况下，通过索引可以更为快速地获取记录，所以对于编码规范、调整良好的数据库，这个等待事件很大通常是正常的。有时候这个等待过高和存储分布不连续、连续数据块中部分被缓存有关，特别对于DML频繁的数据表，数据以及存储空间的不连续可能导致过量的单块读，定期的数据整理和空间回收有时候是必须的。

需要注意在很多情况下，使用索引并不是最佳的选择，比如读取较大表中大量的数据，全表扫描可能会明显快于索引扫描，所以在开发中就应该注意，对于这样的查询应该进行避免使用索引扫描。

16)  Direct Path Read：一般直接路径读取是指将数据块直接读入PGA中。一般用于排序、并行查询和read ahead操作。这个等待可能是由于I/O造成的。使用异步I/O模式或者限制排序在磁盘上，可能会降低这里的等待时间。

与直接读取相关联的等待事件。当ORACLE将数据块直接读入会话的PGA（进程全局区）中，同时绕过SGA（系统全局区）。PGA中的数据并不和其他的会话共享。即表明，读入的这部分数据该会话独自使用，不放于共享的SGA中。

在排序操作(order by/group by/union/distinct/rollup/合并连接)时，由于PGA中的SORT\_AREA\_SIZE空间不足，造成需要使用临时表空间来保存中间结果，当从临时表空间读入排序结果时，产生direct path read等待事件。

使用HASH连接的SQL语句，将不适合位于内存中的散列分区刷新到临时表空间中。为了查明匹配SQL谓词的行，临时表空间中的散列分区被读回到内存中(目的是为了查明匹配SQL谓词的行)，ORALCE会话在direct path read等待事件上等待。

使用并行扫描的SQL语句也会影响系统范围的direct path read等待事件。在并行执行过程中，direct path read等待事件与从属查询有关，而与父查询无关，运行父查询的会话基本上会在PX Deq:Execute Reply上等待，从属查询会产生direct path read等待事件。

直接读取可能按照同步或异步的方式执行，取决于平台和初始化参数disk\_asynch\_io参数的值。使用异步I/O时，系统范围的等待的事件的统计可能不准确，会造成误导作用。

17)  direct path write：直接路径写该等待发生在，系统等待确认所有未完成的异步I/O 都已写入磁盘。对于这一写入等待，我们应该找到I/O 操作最为频繁的数据文件(如果有过多的排序操作，很有可能就是临时文件)，分散负载，加快其写入操作。如果系统存在过多的磁盘排序，会导致临时表空间操作频繁，对于这种情况，可以考虑使用Local管理表空间，分成多个小文件，写入不同磁盘或者裸设备。

在DSS系统中，存在大量的direct path read是很正常的，但是在OLTP系统中，通常显著的直接路径读（direct path read）都意味着系统应用存在问题，从而导致大量的磁盘排序读取操作。

直接路径写（direct paht write）通常发生在Oracle直接从PGA写数据到数据文件或临时文件，这个写操作可以绕过SGA。

这类写入操作通常在以下情况被使用：  
·直接路径加载；  
·并行DML操作；  
·磁盘排序；  
·对未缓存的“LOB”段的写入，随后会记录为direct path write(lob)等待。

最为常见的直接路径写，多数因为磁盘排序导致。对于这一写入等待，我们应该找到I/O操作最为频繁的数据文件（如果有过多的排序操作，很有可能就是临时文件），分散负载，加快其写入操作。

18)  control file parallel write：当server 进程更新所有控制文件时，这个事件可能出现。如果等待很短，可以不用考虑。如果等待时间较长，检查存放控制文件的物理磁盘I/O 是否存在瓶颈。  
多个控制文件是完全相同的拷贝，用于镜像以提高安全性。对于业务系统，多个控制文件应该存放在不同的磁盘上，一般来说三个是足够的，如果只有两个物理硬盘，那么两个控制文件也是可以接受的。在同一个磁盘上保存多个控制文件是不具备实际意义的。减少这个等待，可以考虑如下方法:①减少控制文件的个数(在确保安全的前提下)。②如果系统支持，使用异步IO。③转移控制文件到IO 负担轻的物理磁盘。

19)  control file sequential read   
control file single write ：控制文件连续读/控制文件单个写对单个控制文件I/O 存在问题时，这两个事件会出现。如果等待比较明显，检查单个控制文件，看存放位置是否存在I/O 瓶颈。

20)   library cache pin

该事件通常是发生在先有会话在运行PL/SQL,VIEW,TYPES等object时,又有另外的会话执行重新编译这些object,即先给对象加上了一个共享锁,然后又给它加排它锁,这样在加排它锁的会话上就会出现这个等待。P1,P2可与x$kglpn和x$kglob表相关   
X$KGLOB (Kernel Generic Library Cache Manager Object)   
X$KGLPN (Kernel Generic Library Cache Manager Object Pins)   
-- 查询X$KGLOB,可找到相关的object,其SQL语句如下   
(即把V$SESSION\_WAIT中的P1raw与X$KGLOB中的KGLHDADR相关连)   
select kglnaown,kglnaobj from X$KGLOB   
where KGLHDADR =(select p1raw from v$session\_wait   
where event='library cache pin')   
-- 查出引起该等待事件的阻塞者的sid   
select sid from x$kglpn , v$session   
where KGLPNHDL in   
(select p1raw from v$session\_wait   
where wait\_time=0 and event like 'library cache pin%')   
and KGLPNMOD <> 0   
and v$session.saddr=x$kglpn.kglpnuse   
-- 查出阻塞者正执行的SQL语句   
select sid,sql\_text   
from v$session, v$sqlarea   
where v$session.sql\_address=v$sqlarea.address   
and sid=<阻塞者的sid>   
这样,就可找到"library cache pin"等待的根源，从而解决由此引起的性能问题。

21)  **library cache lock**  
该事件通常是由于执行多个DDL操作导致的,即在library cache object上添加一个排它锁后,又从另一个会话给它添加一个排它锁,这样在第二个会话就会生成等待。可通过到基表x$kgllk中查找其对应的对象。   
-- 查询引起该等待事件的阻塞者的sid、会话用户、锁住的对象   
select b.sid,a.user\_name,a.kglnaobj   
from x$kgllk a , v$session b   
where a.kgllkhdl in   
(select p1raw from v$session\_wait   
where wait\_time=0 and event = 'library cache lock')   
and a.kgllkmod <> 0   
and b.saddr=a.kgllkuse   
当然也可以直接从v$locked\_objects中查看，但没有上面语句直观根据sid可以到v$process中查出pid，然后将其kill或者其它处理。

22)

对于常见的一些IDLE wait事件举例：

dispatcher timer

lock element cleanup

Null event

parallel query dequeue wait

parallel query idle wait - Slaves

pipe get

PL/SQL lock timer

pmon timer- pmon

rdbms ipc message

slave wait

smon timer

SQL\*Net break/reset to client

SQL\*Net message from client

SQL\*Net message to client

SQL\*Net more data to client

virtual circuit status

client message

SQL\*Net message from client

下面是关于这里的常见的等待事件和解决方法的一个快速预览

|  |  |
| --- | --- |
| 等待事件 | 一般解决方法 |
| Sequential Read | 调整相关的索引和选择合适的驱动行源 |
| Scattered Read | 表明出现很多全表扫描。优化code，cache小表到内存中。 |
| Free Buffer | 增大DB\_CACHE\_SIZE，增大checkpoint的频率，优化代码 |
| Buffer Busy Segment header | 增加freelist或者freelistgroups |
| Buffer Busy Data block | 隔离热块；使用反转索引；使用更小的块；增大表的initrans |
| Buffer Busy Undo header | 增加回滚段的数量或者大小 |
| Buffer Busy Undo block | Commit more；增加回滚段的数量或者大小 |
| Latch Free | 检查具体的等待latch类型，解决方法参考后面介绍 |
| Enqueue–ST | 使用本地管理的表空间或者增加预分配的盘区大小 |
| Enqueue–HW | 在HWM之上预先分配盘区 |
| Enqueue–TX4 | 在表或者索引上增大initrans的值或者使用更小的块 |
| Log Buffer Space | 增大LOG\_BUFFER，改善I/O |
| Log File Switch | 增加或者增大日志文件 |
| Log file sync | 减小提交的频率；使用更快的I/O；或者使用裸设备 |
| Write complete waits | 增加DBWR；提高CKPT的频率； |

### Time Model Statistics

* **Total time in database user-calls (DB Time): 663s**
* **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** |
| DB CPU | 514.50 | 77.61 |
| sql execute elapsed time | 482.27 | 72.74 |
| parse time elapsed | 3.76 | 0.57 |
| PL/SQL execution elapsed time | 0.50 | 0.08 |
| hard parse elapsed time | 0.34 | 0.05 |
| connection management call elapsed time | 0.08 | 0.01 |
| hard parse (sharing criteria) elapsed time | 0.00 | 0.00 |
| repeated bind elapsed time | 0.00 | 0.00 |
| PL/SQL compilation elapsed time | 0.00 | 0.00 |
| failed parse elapsed time | 0.00 | 0.00 |
| DB time | 662.97 |  |
| background elapsed time | 185.19 |  |
| background cpu time | 67.48 |  |

此节显示了各种类型的数据库处理任务所占用的CPU时间。

DB time=报表头部显示的db time=cpu time + all of nonidle wait event time

[**Back to Wait Events Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#21)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Wait Class 等待事件的类型

* **s - second**
* **cs - centisecond - 100th of a second**
* **ms - millisecond - 1000th of a second**
* **us - microsecond - 1000000th of a second**
* **ordered by wait time desc, waits desc**

查询Oracle 10gR1提供的12个等待事件类：

select wait\_class#, wait\_class\_id, wait\_class from v$event\_name group by wait\_class#, wait\_class\_id, wait\_class order by wait\_class#;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Wait Class** | **Waits** | **%Time -outs** | **Total Wait Time (s)** | **Avg wait (ms)** | **Waits /txn** |
| User I/O | 66,837 | 0.00 | 120 | 2 | 11.94 |
| System I/O | 28,295 | 0.00 | 93 | 3 | 5.05 |
| Network | 1,571,450 | 0.00 | 66 | 0 | 280.72 |
| Cluster | 210,548 | 0.00 | 29 | 0 | 37.61 |
| Other | 81,783 | 71.82 | 28 | 0 | 14.61 |
| Application | 333,155 | 0.00 | 16 | 0 | 59.51 |
| Concurrency | 5,182 | 0.04 | 5 | 1 | 0.93 |
| Commit | 919 | 0.00 | 4 | 4 | 0.16 |
| Configuration | 25,427 | 99.46 | 1 | 0 | 4.54 |

[**Back to Wait Events Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#21)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Wait Events 现实非空闲等待事件 后面是空闲等待事件

* **s - second**
* **cs - centisecond - 100th of a second**
* **ms - millisecond - 1000th of a second**
* **us - microsecond - 1000000th of a second**
* **ordered by wait time desc, waits desc (idle events last)**

（1）查询所有等待事件及其属性：

select event#, name, parameter1, parameter2, parameter3 from v$event\_name order by name;

（2）查询Oracle 10gR1提供的12个等待事件类：

select wait\_class#, wait\_class\_id, wait\_class from v$event\_name group by wait\_class#, wait\_class\_id, wait\_class order by wait\_class#;

wait\_event.doc

**下面显示的内容可能来自下面几个视图）**

V$EVENT\_NAME视图包含所有为数据库实例定义的等待事件。

V$SYSTEM\_EVENT视图显示自从实例启动后，所有Oracle会话遇到的所有等待事件的总计统计。

V$SESSION\_EVENT视图包含当前连接到实例的所有会话的总计等待事件统计。该视图包含了V$SYSTEM\_EVENT视图中出现的所有列。它记录会话中每一个等待事件的总等待次数、已等待时间和最大等待时间。SID列标识出独立的会话。每个会话中每个事件的最大等待时间在MAX\_WAIT列中追踪。通过用SID列将V$SESSION\_EVENT视图和V$SESSION视图结合起来，可得到有关会话和用户的更多信息。

V$SESSION\_WAIT视图提供关于每个会话正在等待的事件或资源的详细信息。该视图在任何给定时间，只包含每个会话的一行活动的或不活动的信息。

自从OWI在Oracle 7.0.12中引入后，就具有下来4个V$视图：

V$EVENT\_NAME

V$SESSION\_WAIT

V$SESSION\_EVENT

V$SYSTEM\_EVENT

除了这些等待事件视图之外，Oracle 10gR1中引入了下列新视图以从多个角度显示等待信息：

V$SYSTEM\_WAIT\_CLASS

V$SESSION\_WAIT\_CLASS

V$SESSION\_WAIT\_HISTORY

V$EVENT\_HISTOGRAM

V$ACTIVE\_SESSION\_HISTORY

然而，V$SESSION\_WAIT、V$SESSION\_WAIT和V$SESSION\_WAIT仍然是3个重要的视图，它们提供了不同粒度级的等待事件统计和计时信息。三者的关系如下：

V$SESSION\_WAIT  V$SESSION\_EVENT  V$SYSTEM\_EVENT

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Event** | **Waits** | **%Time -outs** | **Total Wait Time (s)** | **Avg wait (ms)** | **Waits /txn** |
| SQL\*Net more data from client | 27,319 | 0.00 | 64 | 2 | 4.88 |
| log file parallel write | 5,497 | 0.00 | 47 | 9 | 0.98 |
| db file sequential read | 7,900 | 0.00 | 35 | 4 | 1.41 |
| db file parallel write | 4,806 | 0.00 | 34 | 7 | 0.86 |
| db file scattered read | 10,310 | 0.00 | 31 | 3 | 1.84 |
| direct path write | 42,724 | 0.00 | 30 | 1 | 7.63 |
| reliable message | 355 | 2.82 | 18 | 49 | 0.06 |
| SQL\*Net break/reset to client | 333,084 | 0.00 | 16 | 0 | 59.50 |
| db file parallel read | 3,732 | 0.00 | 13 | 4 | 0.67 |
| gc current multi block request | 175,710 | 0.00 | 10 | 0 | 31.39 |
| control file sequential read | 15,974 | 0.00 | 10 | 1 | 2.85 |
| direct path read temp | 1,873 | 0.00 | 9 | 5 | 0.33 |
| gc cr multi block request | 20,877 | 0.00 | 8 | 0 | 3.73 |
| log file sync | 919 | 0.00 | 4 | 4 | 0.16 |
| gc cr block busy | 526 | 0.00 | 3 | 6 | 0.09 |
| enq: FB - contention | 10,384 | 0.00 | 3 | 0 | 1.85 |
| DFS lock handle | 3,517 | 0.00 | 3 | 1 | 0.63 |
| control file parallel write | 1,946 | 0.00 | 3 | 1 | 0.35 |
| gc current block 2-way | 4,165 | 0.00 | 2 | 0 | 0.74 |
| library cache lock | 432 | 0.00 | 2 | 4 | 0.08 |
| name-service call wait | 22 | 0.00 | 2 | 76 | 0.00 |
| row cache lock | 3,894 | 0.00 | 2 | 0 | 0.70 |
| gcs log flush sync | 1,259 | 42.02 | 2 | 1 | 0.22 |
| os thread startup | 18 | 5.56 | 2 | 89 | 0.00 |
| gc cr block 2-way | 3,671 | 0.00 | 2 | 0 | 0.66 |
| gc current block busy | 113 | 0.00 | 1 | 12 | 0.02 |
| SQL\*Net message to client | 1,544,115 | 0.00 | 1 | 0 | 275.83 |
| gc buffer busy | 15 | 6.67 | 1 | 70 | 0.00 |
| gc cr disk read | 3,272 | 0.00 | 1 | 0 | 0.58 |
| direct path write temp | 159 | 0.00 | 1 | 5 | 0.03 |
| gc current grant busy | 898 | 0.00 | 1 | 1 | 0.16 |
| log file switch completion | 29 | 0.00 | 1 | 17 | 0.01 |
| CGS wait for IPC msg | 48,739 | 99.87 | 0 | 0 | 8.71 |
| gc current grant 2-way | 1,142 | 0.00 | 0 | 0 | 0.20 |
| kjbdrmcvtq lmon drm quiesce: ping completion | 9 | 0.00 | 0 | 19 | 0.00 |
| enq: US - contention | 567 | 0.00 | 0 | 0 | 0.10 |
| direct path read | 138 | 0.00 | 0 | 1 | 0.02 |
| enq: WF - contention | 14 | 0.00 | 0 | 9 | 0.00 |
| ksxr poll remote instances | 13,291 | 58.45 | 0 | 0 | 2.37 |
| library cache pin | 211 | 0.00 | 0 | 1 | 0.04 |
| ges global resource directory to be frozen | 9 | 100.00 | 0 | 10 | 0.00 |
| wait for scn ack | 583 | 0.00 | 0 | 0 | 0.10 |
| log file sequential read | 36 | 0.00 | 0 | 2 | 0.01 |
| undo segment extension | 25,342 | 99.79 | 0 | 0 | 4.53 |
| rdbms ipc reply | 279 | 0.00 | 0 | 0 | 0.05 |
| ktfbtgex | 6 | 100.00 | 0 | 10 | 0.00 |
| enq: HW - contention | 44 | 0.00 | 0 | 1 | 0.01 |
| gc cr grant 2-way | 158 | 0.00 | 0 | 0 | 0.03 |
| enq: TX - index contention | 1 | 0.00 | 0 | 34 | 0.00 |
| enq: CF - contention | 64 | 0.00 | 0 | 1 | 0.01 |
| PX Deq: Signal ACK | 37 | 21.62 | 0 | 1 | 0.01 |
| latch free | 3 | 0.00 | 0 | 10 | 0.00 |
| buffer busy waits | 625 | 0.16 | 0 | 0 | 0.11 |
| KJC: Wait for msg sends to complete | 154 | 0.00 | 0 | 0 | 0.03 |
| log buffer space | 11 | 0.00 | 0 | 2 | 0.00 |
| enq: PS - contention | 46 | 0.00 | 0 | 1 | 0.01 |
| enq: TM - contention | 70 | 0.00 | 0 | 0 | 0.01 |
| IPC send completion sync | 40 | 100.00 | 0 | 0 | 0.01 |
| PX Deq: reap credit | 1,544 | 99.81 | 0 | 0 | 0.28 |
| log file single write | 36 | 0.00 | 0 | 0 | 0.01 |
| enq: TT - contention | 46 | 0.00 | 0 | 0 | 0.01 |
| enq: TD - KTF dump entries | 12 | 0.00 | 0 | 1 | 0.00 |
| read by other session | 1 | 0.00 | 0 | 12 | 0.00 |
| LGWR wait for redo copy | 540 | 0.00 | 0 | 0 | 0.10 |
| PX Deq Credit: send blkd | 17 | 5.88 | 0 | 0 | 0.00 |
| enq: TA - contention | 14 | 0.00 | 0 | 0 | 0.00 |
| latch: ges resource hash list | 44 | 0.00 | 0 | 0 | 0.01 |
| enq: PI - contention | 8 | 0.00 | 0 | 0 | 0.00 |
| write complete waits | 1 | 0.00 | 0 | 2 | 0.00 |
| enq: DR - contention | 3 | 0.00 | 0 | 0 | 0.00 |
| enq: MW - contention | 3 | 0.00 | 0 | 0 | 0.00 |
| enq: TS - contention | 3 | 0.00 | 0 | 0 | 0.00 |
| PX qref latch | 150 | 100.00 | 0 | 0 | 0.03 |
| PX qref latch  在并行执行的情况下偶然会发现PX qref latch等待事件，当系统高峰期同时采用了高并发的情况下最容易出现。看来要进行特殊照顾了。  **概念和原理** 在并行执行环境中，query slaves 和query coordinator之间是通过队列交换数据和信息的。PX qref latch 是用来保护这些队列的。 PX qref latch 等待事件的出现一般表明信息的发送比接受快，这时需要调整buffer size（可以通过parallel\_execution\_message\_size参数调整）。 但是有些情况下也是难以避免发生这种情况的，比如consumer需要长时间的等待数据的处理，原因在于需要返回大批量的数据包，这种情况下很正常。  **调整和措施** 当系统的负载比较高时，需要把并行度降低；如果使用的是默认并行度，可以通过减小parallel\_thread\_per\_cpu参数的值来达到效果。 DEFAULT degree = PARALLEL\_THREADS\_PER\_CPU \* #CPU's  **优化parallel\_execution\_message\_size参数** Tuning parallel\_execution\_message\_size is a tradeoff between performance and memory. For parallel query, the connection topology between slaves and QC requires (n^2 + 2n) connections (where n is the DOP not the actual number of slaves) at maximum.  If each connection has 3 buffers associated with it then you can very quickly get into high memory consumption on large machines  doing high DOP queries | | | | | |
| enq: MD - contention | 2 | 0.00 | 0 | 0 | 0.00 |
| latch: KCL gc element parent latch | 11 | 0.00 | 0 | 0 | 0.00 |
| enq: JS - job run lock - synchronize | 1 | 0.00 | 0 | 1 | 0.00 |
| SQL\*Net more data to client | 16 | 0.00 | 0 | 0 | 0.00 |
| latch: cache buffers lru chain | 1 | 0.00 | 0 | 0 | 0.00 |
| enq: UL - contention | 1 | 0.00 | 0 | 0 | 0.00 |
| gc current split | 1 | 0.00 | 0 | 0 | 0.00 |
| enq: AF - task serialization | 1 | 0.00 | 0 | 0 | 0.00 |
| latch: object queue header operation | 3 | 0.00 | 0 | 0 | 0.00 |
| latch: cache buffers chains | 1 | 0.00 | 0 | 0 | 0.00 |
| latch: enqueue hash chains | 2 | 0.00 | 0 | 0 | 0.00 |
| SQL\*Net message from client | 1,544,113 | 0.00 | 12,626 | 8 | 275.83 |
| gcs remote message | 634,884 | 98.64 | 9,203 | 14 | 113.41 |
| DIAG idle wait | 23,628 | 0.00 | 4,616 | 195 | 4.22 |
| ges remote message | 149,591 | 93.45 | 4,612 | 31 | 26.72 |
| Streams AQ: qmn slave idle wait | 167 | 0.00 | 4,611 | 27611 | 0.03 |
| Streams AQ: qmn coordinator idle wait | 351 | 47.86 | 4,611 | 13137 | 0.06 |
| Streams AQ: waiting for messages in the queue | 488 | 100.00 | 4,605 | 9436 | 0.09 |
| virtual circuit status | 157 | 100.00 | 4,596 | 29272 | 0.03 |
| PX Idle Wait | 1,072 | 97.11 | 2,581 | 2407 | 0.19 |
| jobq slave wait | 145 | 97.93 | 420 | 2896 | 0.03 |
| Streams AQ: waiting for time management or cleanup tasks | 1 | 100.00 | 270 | 269747 | 0.00 |
| PX Deq: Parse Reply | 40 | 40.00 | 0 | 3 | 0.01 |
| PX Deq: Execution Msg | 121 | 26.45 | 0 | 0 | 0.02 |
| PX Deq: Join ACK | 38 | 42.11 | 0 | 1 | 0.01 |
| PX Deq: Execute Reply | 34 | 32.35 | 0 | 0 | 0.01 |
| PX Deq: Msg Fragment | 16 | 0.00 | 0 | 0 | 0.00 |
| Streams AQ: RAC qmn coordinator idle wait | 351 | 100.00 | 0 | 0 | 0.06 |
| class slave wait | 2 | 0.00 | 0 | 0 | 0.00 |

db file scattered read等待事件是当SESSION等待multi-block I/O时发生的，通过是由于full table scans或 index fast full scans。发生过多读操作的Segments可以在“Segments by Physical Reads”和 “SQL ordered by Reads”节中识别（在其它版本的报告中，可能是别的名称）。如果在OLTP应用中，不应该有过多的全扫描操作，而应使用选择性好的索引操作。

DB file sequential read等待意味着发生顺序I/O读等待（通常是单块读取到连续的内存区域中），如果这个等待非常严重，应该使用上一段的方法确定执行读操作的热点SEGMENT，然后通过对大表进行分区以减少I/O量，或者优化执行计划（通过使用存储大纲或执行数据分析）以避免单块读操作引起的sequential read等待。通过在批量应用中，DB file sequential read是很影响性能的事件，总是应当设法避免。

Log File Parallel Write 事件是在等待LGWR进程将REDO记录从LOG 缓冲区写到联机日志文件时发生的。虽然写操作可能是并发的，但LGWR需要等待最后的I/O写到磁盘上才能认为并行写的完成，因此等待时间依赖于OS完成所有请求的时间。如果这个等待比较严重，可以通过将LOG文件移到更快的磁盘上或者条带化磁盘（减少争用）而降低这个等待。

Buffer Busy Waits事件是在一个SESSION需要访问BUFFER CACHE中的一个数据库块而又不能访问时发生的。缓冲区“busy”的两个原因是：1）另一个SESSION正在将数据块读进BUFFER。2）另一个SESSION正在以排它模式占用着这块被请求的BUFFER。可以在“Segments by Buffer Busy Waits”一节中找出发生这种等待的SEGMENT，然后通过使用reverse-key indexes并对热表进行分区而减少这种等待事件。

Log File Sync事件，当用户SESSION执行事务操作（COMMIT或ROLLBACK等）后，会通知 LGWR进程将所需要的所有REDO信息从LOG BUFFER写到LOG文件，在用户SESSION等待LGWR返回安全写入磁盘的通知时发生此等待。减少此等待的方法写Log File Parallel Write事件的处理。

Enqueue Waits是串行访问本地资源的本锁，表明正在等待一个被其它SESSION（一个或多个）以排它模式锁住的资源。减少这种等待的方法依赖于生产等待的锁类型。导致Enqueue等待的主要锁类型有三种：TX（事务锁）, TM D（ML锁）和ST（空间管理锁）。

[**Back to Wait Events Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#21)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Background Wait Events

* **ordered by wait time desc, waits desc (idle events last)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Event** | **Waits** | **%Time -outs** | **Total Wait Time (s)** | **Avg wait (ms)** | **Waits /txn** |
| log file parallel write | 5,497 | 0.00 | 47 | 9 | 0.98 |
| db file parallel write | 4,806 | 0.00 | 34 | 7 | 0.86 |
| events in waitclass Other | 69,002 | 83.25 | 22 | 0 | 12.33 |
| control file sequential read | 9,323 | 0.00 | 7 | 1 | 1.67 |
| control file parallel write | 1,946 | 0.00 | 3 | 1 | 0.35 |
| os thread startup | 18 | 5.56 | 2 | 89 | 0.00 |
| direct path read | 138 | 0.00 | 0 | 1 | 0.02 |
| db file sequential read | 21 | 0.00 | 0 | 5 | 0.00 |
| direct path write | 138 | 0.00 | 0 | 0 | 0.02 |
| log file sequential read | 36 | 0.00 | 0 | 2 | 0.01 |
| gc cr block 2-way | 96 | 0.00 | 0 | 0 | 0.02 |
| gc current block 2-way | 78 | 0.00 | 0 | 0 | 0.01 |
| log buffer space | 11 | 0.00 | 0 | 2 | 0.00 |
| row cache lock | 59 | 0.00 | 0 | 0 | 0.01 |
| log file single write | 36 | 0.00 | 0 | 0 | 0.01 |
| buffer busy waits | 151 | 0.66 | 0 | 0 | 0.03 |
| gc current grant busy | 29 | 0.00 | 0 | 0 | 0.01 |
| library cache lock | 4 | 0.00 | 0 | 1 | 0.00 |
| enq: TM - contention | 10 | 0.00 | 0 | 0 | 0.00 |
| gc current grant 2-way | 8 | 0.00 | 0 | 0 | 0.00 |
| gc cr multi block request | 7 | 0.00 | 0 | 0 | 0.00 |
| gc cr grant 2-way | 5 | 0.00 | 0 | 0 | 0.00 |
| rdbms ipc message | 97,288 | 73.77 | 50,194 | 516 | 17.38 |
| gcs remote message | 634,886 | 98.64 | 9,203 | 14 | 113.41 |
| DIAG idle wait | 23,628 | 0.00 | 4,616 | 195 | 4.22 |
| pmon timer | 1,621 | 100.00 | 4,615 | 2847 | 0.29 |
| ges remote message | 149,591 | 93.45 | 4,612 | 31 | 26.72 |
| Streams AQ: qmn slave idle wait | 167 | 0.00 | 4,611 | 27611 | 0.03 |
| Streams AQ: qmn coordinator idle wait | 351 | 47.86 | 4,611 | 13137 | 0.06 |
| smon timer | 277 | 6.50 | 4,531 | 16356 | 0.05 |
| Streams AQ: waiting for time management or cleanup tasks | 1 | 100.00 | 270 | 269747 | 0.00 |
| PX Deq: Parse Reply | 40 | 40.00 | 0 | 3 | 0.01 |
| PX Deq: Join ACK | 38 | 42.11 | 0 | 1 | 0.01 |
| PX Deq: Execute Reply | 34 | 32.35 | 0 | 0 | 0.01 |
| Streams AQ: RAC qmn coordinator idle wait | 351 | 100.00 | 0 | 0 | 0.06 |

[**Back to Wait Events Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#21)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Operating System Statistics

|  |  |
| --- | --- |
| **Statistic** | **Total** |
| NUM\_LCPUS | 0 |
| NUM\_VCPUS | 0 |
| AVG\_BUSY\_TIME | 101,442 |
| AVG\_IDLE\_TIME | 371,241 |
| AVG\_IOWAIT\_TIME | 5,460 |
| AVG\_SYS\_TIME | 25,795 |
| AVG\_USER\_TIME | 75,510 |
| BUSY\_TIME | 812,644 |
| IDLE\_TIME | 2,971,077 |
| IOWAIT\_TIME | 44,794 |
| SYS\_TIME | 207,429 |
| USER\_TIME | 605,215 |
| LOAD | 0 |
| OS\_CPU\_WAIT\_TIME | 854,100 |
| RSRC\_MGR\_CPU\_WAIT\_TIME | 0 |
| PHYSICAL\_MEMORY\_BYTES | 8,589,934,592 |
| NUM\_CPUS | 8 |
| NUM\_CPU\_CORES | 4 |

NUM\_LCPUS：                  如果显示0，是因为没有设置LPARS

NUM\_VCPUS：                    同上。

AVG\_BUSY\_TIME：           BUSY\_TIME / NUM\_CPUS

AVG\_IDLE\_TIME：             IDLE\_TIME / NUM\_CPUS

AVG\_IOWAIT\_TIME：              IOWAIT\_TIME / NUM\_CPUS

AVG\_SYS\_TIME：               SYS\_TIME / NUM\_CPUS

AVG\_USER\_TIME：            USER\_TIME / NUM\_CPUSar o

BUSY\_TIME：                      time equiv of %usr+%sys in sar output

IDLE\_TIME：                        time equiv of %idle in sar

IOWAIT\_TIME：                  time equiv of %wio in sar

SYS\_TIME：                          time equiv of %sys in sar

USER\_TIME：                       time equiv of %usr in sar

LOAD：                                  未知

OS\_CPU\_WAIT\_TIME：      supposedly time waiting on run queues

RSRC\_MGR\_CPU\_WAIT\_TIME：   time waited coz of resource manager

PHYSICAL\_MEMORY\_BYTES：    total memory in use supposedly

NUM\_CPUS：                       number of CPUs reported by OS 操作系统CPU数

NUM\_CPU\_CORES：          number of CPU sockets on motherboard 主板上CPU插槽数

总的elapsed time也可以用以公式计算：

BUSY\_TIME + IDLE\_TIME + IOWAIT TIME

或：SYS\_TIME + USER\_TIME + IDLE\_TIME + IOWAIT\_TIME

 （因为BUSY\_TIME = SYS\_TIME+USER\_TIME）

[**Back to Wait Events Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#21)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Service Statistics

* **ordered by DB Time**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Service Name** | **DB Time (s)** | **DB CPU (s)** | **Physical Reads** | **Logical Reads** |
| ICCI | 608.10 | 496.60 | 315,849 | 16,550,972 |
| SYS$USERS | 54.70 | 17.80 | 6,539 | 58,929 |
| ICCIXDB | 0.00 | 0.00 | 0 | 0 |
| SYS$BACKGROUND | 0.00 | 0.00 | 282 | 38,990 |

[**Back to Wait Events Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#21)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Service Wait Class Stats

* **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 centisecond (100th of a second)**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Service Name** | **User I/O Total Wts** | **User I/O Wt Time** | **Concurcy Total Wts** | **Concurcy Wt Time** | **Admin Total Wts** | **Admin Wt Time** | **Network Total Wts** | **Network Wt Time** |
| ICCI | 59826 | 8640 | 4621 | 338 | 0 | 0 | 1564059 | 6552 |
| SYS$USERS | 6567 | 3238 | 231 | 11 | 0 | 0 | 7323 | 3 |
| SYS$BACKGROUND | 443 | 115 | 330 | 168 | 0 | 0 | 0 | 0 |

[**Back to Wait Events Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#21)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

## SQL Statistics v$sqlarea

* [**SQL ordered by Elapsed Time**](http://blog.itpub.net/26736162/viewspace-1869086/#400)
* [**SQL ordered by CPU Time**](http://blog.itpub.net/26736162/viewspace-1869086/#500)
* [**SQL ordered by Gets**](http://blog.itpub.net/26736162/viewspace-1869086/#600)
* [**SQL ordered by Reads**](http://blog.itpub.net/26736162/viewspace-1869086/#700)
* [**SQL ordered by Executions**](http://blog.itpub.net/26736162/viewspace-1869086/#800)
* [**SQL ordered by Parse Calls**](http://blog.itpub.net/26736162/viewspace-1869086/#900)
* [**SQL ordered by Sharable Memory**](http://blog.itpub.net/26736162/viewspace-1869086/#1000)
* [**SQL ordered by Version Count**](http://blog.itpub.net/26736162/viewspace-1869086/#1100)
* [**SQL ordered by Cluster Wait Time**](http://blog.itpub.net/26736162/viewspace-1869086/#1150)
* [**Complete List of SQL Text**](http://blog.itpub.net/26736162/viewspace-1869086/#99997)

本节按各种资源分别列出对资源消耗最严重的SQL语句，并显示它们所占统计期内全部资源的比例，这给出我们调优指南。例如在一个系统中，CPU资源是系统性能瓶颈所在，那么优化buffer gets最多的SQL语句将获得最大效果。在一个I/O等待是最严重事件的系统中，调优的目标应该是physical IOs最多的SQL语句。

在STATSPACK报告中，没有完整的SQL语句，可使用报告中的Hash Value通过下面语句从数据库中查到：

select sql\_text

from stats$sqltext

where hash\_value = &hash\_value

order by piece;

[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### SQL ordered by Elapsed Time

* **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**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Elapsed Time (s)** | **CPU Time (s)** | **Executions** | **Elap per Exec (s)** | **% Total DB Time** | **SQL Id** | **SQL Module** | **SQL Text** |
| 93 | 57 | 1 | 93.50 | 14.10 | [**d8z0u8hgj8xdy**](http://blog.itpub.net/26736162/viewspace-1869086/#d8z0u8hgj8xdy) | cuidmain@HPGICCI1 (TNS V1-V3) | insert into CUID select CUID\_... |
| 76 | 75 | 172,329 | 0.00 | 11.52 | [**4vja2k2gdtyup**](http://blog.itpub.net/26736162/viewspace-1869086/#4vja2k2gdtyup) | load\_fnsact@HPGICCI1 (TNS V1-V3) | insert into ICCICCS values (:... |
| 58 | 42 | 1 | 58.04 | 8.75 | [**569r5k05drsj7**](http://blog.itpub.net/26736162/viewspace-1869086/#569r5k05drsj7) | cumimain@HPGICCI1 (TNS V1-V3) | insert into CUMI select CUSV\_... |
| 51 | 42 | 1 | 50.93 | 7.68 | [**ackxqhnktxnbc**](http://blog.itpub.net/26736162/viewspace-1869086/#ackxqhnktxnbc) | cusmmain@HPGICCI1 (TNS V1-V3) | insert into CUSM select CUSM\_... |
| 38 | 36 | 166,069 | 0.00 | 5.67 | [**7gtztzv329wg0**](http://blog.itpub.net/26736162/viewspace-1869086/#7gtztzv329wg0) |  | select c.name, u.name from co... |
| 35 | 3 | 1 | 35.00 | 5.28 | [**6z06gcfw39pkd**](http://blog.itpub.net/26736162/viewspace-1869086/#6z06gcfw39pkd) | SQL\*Plus | SELECT F.TABLESPACE\_NAME, TO\_... |
| 23 | 23 | 172,329 | 0.00 | 3.46 | [**1dm3bq36vu3g8**](http://blog.itpub.net/26736162/viewspace-1869086/#1dm3bq36vu3g8) | load\_fnsact@HPGICCI1 (TNS V1-V3) | insert into iccifnsact values... |
| 15 | 11 | 5 | 2.98 | 2.25 | [**djs2w2f17nw2z**](http://blog.itpub.net/26736162/viewspace-1869086/#djs2w2f17nw2z) |  | DECLARE job BINARY\_INTEGER := ... |
| 14 | 14 | 172,983 | 0.00 | 2.16 | [**7wwv1ybs9zguz**](http://blog.itpub.net/26736162/viewspace-1869086/#7wwv1ybs9zguz) | load\_fnsact@HPGICCI1 (TNS V1-V3) | update ICCIFNSACT set BORM\_AD... |
| 13 | 13 | 172,337 | 0.00 | 2.00 | [**gmn2w09rdxn14**](http://blog.itpub.net/26736162/viewspace-1869086/#gmn2w09rdxn14) | load\_oldnewact@HPGICCI1 (TNS V1-V3) | insert into OLDNEWACT values ... |
| 13 | 13 | 166,051 | 0.00 | 1.89 | [**chjmy0dxf9mbj**](http://blog.itpub.net/26736162/viewspace-1869086/#chjmy0dxf9mbj) | icci\_migact@HPGICCI1 (TNS V1-V3) | insert into ICCICCS values (:... |
| 10 | 4 | 1 | 9.70 | 1.46 | [**0yv9t4qb1zb2b**](http://blog.itpub.net/26736162/viewspace-1869086/#0yv9t4qb1zb2b) | cuidmain@HPGICCI1 (TNS V1-V3) | select CUID\_CUST\_NO , CUID\_ID\_... |
| 10 | 8 | 5 | 1.91 | 1.44 | [**1crajpb7j5tyz**](http://blog.itpub.net/26736162/viewspace-1869086/#1crajpb7j5tyz) |  | INSERT INTO STATS$SGA\_TARGET\_A... |
| 8 | 8 | 172,329 | 0.00 | 1.25 | [**38apjgr0p55ns**](http://blog.itpub.net/26736162/viewspace-1869086/#38apjgr0p55ns) | load\_fnsact@HPGICCI1 (TNS V1-V3) | update ICCICCS set CCSMAXOVER... |
| 8 | 8 | 172,983 | 0.00 | 1.16 | [**5c4qu2zmj3gux**](http://blog.itpub.net/26736162/viewspace-1869086/#5c4qu2zmj3gux) | load\_fnsact@HPGICCI1 (TNS V1-V3) | select \* from ICCIPRODCODE wh... |

[**Back to SQL Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#22)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### SQL ordered by CPU Time

* **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**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **CPU Time (s)** | **Elapsed Time (s)** | **Executions** | **CPU per Exec (s)** | **% Total DB Time** | **SQL Id** | **SQL Module** | **SQL Text** |
| 75 | 76 | 172,329 | 0.00 | 11.52 | [**4vja2k2gdtyup**](http://blog.itpub.net/26736162/viewspace-1869086/#4vja2k2gdtyup) | load\_fnsact@HPGICCI1 (TNS V1-V3) | insert into ICCICCS values (:... |
| 57 | 93 | 1 | 57.31 | 14.10 | [**d8z0u8hgj8xdy**](http://blog.itpub.net/26736162/viewspace-1869086/#d8z0u8hgj8xdy) | cuidmain@HPGICCI1 (TNS V1-V3) | insert into CUID select CUID\_... |
| 42 | 51 | 1 | 42.43 | 7.68 | [**ackxqhnktxnbc**](http://blog.itpub.net/26736162/viewspace-1869086/#ackxqhnktxnbc) | cusmmain@HPGICCI1 (TNS V1-V3) | insert into CUSM select CUSM\_... |
| 42 | 58 | 1 | 42.01 | 8.75 | [**569r5k05drsj7**](http://blog.itpub.net/26736162/viewspace-1869086/#569r5k05drsj7) | cumimain@HPGICCI1 (TNS V1-V3) | insert into CUMI select CUSV\_... |
| 36 | 38 | 166,069 | 0.00 | 5.67 | [**7gtztzv329wg0**](http://blog.itpub.net/26736162/viewspace-1869086/#7gtztzv329wg0) |  | select c.name, u.name from co... |
| 23 | 23 | 172,329 | 0.00 | 3.46 | [**1dm3bq36vu3g8**](http://blog.itpub.net/26736162/viewspace-1869086/#1dm3bq36vu3g8) | load\_fnsact@HPGICCI1 (TNS V1-V3) | insert into iccifnsact values... |
| 14 | 14 | 172,983 | 0.00 | 2.16 | [**7wwv1ybs9zguz**](http://blog.itpub.net/26736162/viewspace-1869086/#7wwv1ybs9zguz) | load\_fnsact@HPGICCI1 (TNS V1-V3) | update ICCIFNSACT set BORM\_AD... |
| 13 | 13 | 172,337 | 0.00 | 2.00 | [**gmn2w09rdxn14**](http://blog.itpub.net/26736162/viewspace-1869086/#gmn2w09rdxn14) | load\_oldnewact@HPGICCI1 (TNS V1-V3) | insert into OLDNEWACT values ... |
| 13 | 13 | 166,051 | 0.00 | 1.89 | [**chjmy0dxf9mbj**](http://blog.itpub.net/26736162/viewspace-1869086/#chjmy0dxf9mbj) | icci\_migact@HPGICCI1 (TNS V1-V3) | insert into ICCICCS values (:... |
| 11 | 15 | 5 | 2.23 | 2.25 | [**djs2w2f17nw2z**](http://blog.itpub.net/26736162/viewspace-1869086/#djs2w2f17nw2z) |  | DECLARE job BINARY\_INTEGER := ... |
| 8 | 8 | 172,329 | 0.00 | 1.25 | [**38apjgr0p55ns**](http://blog.itpub.net/26736162/viewspace-1869086/#38apjgr0p55ns) | load\_fnsact@HPGICCI1 (TNS V1-V3) | update ICCICCS set CCSMAXOVER... |
| 8 | 10 | 5 | 1.60 | 1.44 | [**1crajpb7j5tyz**](http://blog.itpub.net/26736162/viewspace-1869086/#1crajpb7j5tyz) |  | INSERT INTO STATS$SGA\_TARGET\_A... |
| 8 | 8 | 172,983 | 0.00 | 1.16 | [**5c4qu2zmj3gux**](http://blog.itpub.net/26736162/viewspace-1869086/#5c4qu2zmj3gux) | load\_fnsact@HPGICCI1 (TNS V1-V3) | select \* from ICCIPRODCODE wh... |
| 4 | 10 | 1 | 3.54 | 1.46 | [**0yv9t4qb1zb2b**](http://blog.itpub.net/26736162/viewspace-1869086/#0yv9t4qb1zb2b) | cuidmain@HPGICCI1 (TNS V1-V3) | select CUID\_CUST\_NO , CUID\_ID\_... |
| 3 | 35 | 1 | 3.13 | 5.28 | [**6z06gcfw39pkd**](http://blog.itpub.net/26736162/viewspace-1869086/#6z06gcfw39pkd) | SQL\*Plus | SELECT F.TABLESPACE\_NAME, TO\_... |

[**Back to SQL Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#22)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### 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: 16,648,792**
* **Captured SQL account for 97.9% of Total**

这一部分，通过Buffer Gets对SQL语句进行排序，即通过它执行了多少个逻辑I/O来排序。顶端的注释表明一个PL/SQL单元的缓存获得(Buffer Gets)包括被这个代码块执行的所有SQL语句的Buffer Gets。因此将经常在这个列表的顶端看到PL/SQL过程，因为存储过程执行的单独的语句的数目被总计出来。在这里的Buffer Gets是一个累积值，所以这个值大并不一定意味着这条语句的性能存在问题。通常我们可以通过对比该条语句的Buffer Gets和physical reads值，如果这两个比较接近，肯定这条语句是存在问题的，我们可以通过执行计划来分析，为什么physical reads的值如此之高。另外，我们在这里也可以关注gets per exec的值，这个值如果太大，表明这条语句可能使用了一个比较差的索引或者使用了不当的表连接。

另外说明一点：大量的逻辑读往往伴随着较高的CPU消耗。所以很多时候我们看到的系统CPU将近100％的时候，很多时候就是SQL语句造成的，这时候我们可以分析一下这里逻辑读大的SQL。

select \* from

          (select substr(sql\_text,1,40) sql,   buffer\_gets,

executions, buffer\_gets/executions "Gets/Exec",

          hash\_value,address

          from v$sqlarea

          where  buffer\_gets > 0 and executions>0

          order by buffer\_gets desc)

          where rownum <= 10 ;

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Buffer Gets** | **Executions** | **Gets per Exec** | **%Total** | **CPU Time (s)** | **Elapsed Time (s)** | **SQL Id** | **SQL Module** | **SQL Text** |
| 3,305,363 | 172,329 | 19.18 | 19.85 | 74.57 | 76.41 | [**4vja2k2gdtyup**](http://blog.itpub.net/26736162/viewspace-1869086/#4vja2k2gdtyup) | load\_fnsact@HPGICCI1 (TNS V1-V3) | insert into ICCICCS values (:... |
| 2,064,414 | 1 | 2,064,414.00 | 12.40 | 57.31 | 93.50 | [**d8z0u8hgj8xdy**](http://blog.itpub.net/26736162/viewspace-1869086/#d8z0u8hgj8xdy) | cuidmain@HPGICCI1 (TNS V1-V3) | insert into CUID select CUID\_... |
| 1,826,869 | 166,069 | 11.00 | 10.97 | 35.84 | 37.60 | [**7gtztzv329wg0**](http://blog.itpub.net/26736162/viewspace-1869086/#7gtztzv329wg0) |  | select c.name, u.name from co... |
| 1,427,648 | 172,337 | 8.28 | 8.58 | 12.97 | 13.29 | [**gmn2w09rdxn14**](http://blog.itpub.net/26736162/viewspace-1869086/#gmn2w09rdxn14) | load\_oldnewact@HPGICCI1 (TNS V1-V3) | insert into OLDNEWACT values ... |
| 1,278,667 | 172,329 | 7.42 | 7.68 | 22.85 | 22.94 | [**1dm3bq36vu3g8**](http://blog.itpub.net/26736162/viewspace-1869086/#1dm3bq36vu3g8) | load\_fnsact@HPGICCI1 (TNS V1-V3) | insert into iccifnsact values... |
| 1,216,367 | 1 | 1,216,367.00 | 7.31 | 42.43 | 50.93 | [**ackxqhnktxnbc**](http://blog.itpub.net/26736162/viewspace-1869086/#ackxqhnktxnbc) | cusmmain@HPGICCI1 (TNS V1-V3) | insert into CUSM select CUSM\_... |
| 1,107,305 | 1 | 1,107,305.00 | 6.65 | 42.01 | 58.04 | [**569r5k05drsj7**](http://blog.itpub.net/26736162/viewspace-1869086/#569r5k05drsj7) | cumimain@HPGICCI1 (TNS V1-V3) | insert into CUMI select CUSV\_... |
| 898,868 | 172,983 | 5.20 | 5.40 | 14.28 | 14.34 | [**7wwv1ybs9zguz**](http://blog.itpub.net/26736162/viewspace-1869086/#7wwv1ybs9zguz) | load\_fnsact@HPGICCI1 (TNS V1-V3) | update ICCIFNSACT set BORM\_AD... |
| 711,450 | 166,051 | 4.28 | 4.27 | 12.52 | 12.55 | [**chjmy0dxf9mbj**](http://blog.itpub.net/26736162/viewspace-1869086/#chjmy0dxf9mbj) | icci\_migact@HPGICCI1 (TNS V1-V3) | insert into ICCICCS values (:... |
| 692,996 | 172,329 | 4.02 | 4.16 | 8.31 | 8.31 | [**38apjgr0p55ns**](http://blog.itpub.net/26736162/viewspace-1869086/#38apjgr0p55ns) | load\_fnsact@HPGICCI1 (TNS V1-V3) | update ICCICCS set CCSMAXOVER... |
| 666,748 | 166,052 | 4.02 | 4.00 | 6.36 | 6.36 | [**7v9dyf5r424yh**](http://blog.itpub.net/26736162/viewspace-1869086/#7v9dyf5r424yh) | icci\_migact@HPGICCI1 (TNS V1-V3) | select NEWACTNO into :b0 from... |
| 345,357 | 172,983 | 2.00 | 2.07 | 7.70 | 7.71 | [**5c4qu2zmj3gux**](http://blog.itpub.net/26736162/viewspace-1869086/#5c4qu2zmj3gux) | load\_fnsact@HPGICCI1 (TNS V1-V3) | select \* from ICCIPRODCODE wh... |
| 231,756 | 51,633 | 4.49 | 1.39 | 5.75 | 5.83 | [**49ms69srnaxzj**](http://blog.itpub.net/26736162/viewspace-1869086/#49ms69srnaxzj) | load\_fnsact@HPGICCI1 (TNS V1-V3) | insert into ICCIRPYV values (... |

[**Back to SQL Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#22)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### SQL ordered by Reads

* **Total Disk Reads: 322,678**
* **Captured SQL account for 66.1% of Total**

这部分通过物理读对SQL语句进行排序。这显示引起大部分对这个系统进行读取活动的SQL，即物理I/O。当我们的系统如果存在I/O瓶颈时，需要关注这里I/O操作比较多的语句。

select \* from

              (select substr(sql\_text,1,40) sql,      disk\_reads,

executions, disk\_reads/executions "Reads/Exec",

              hash\_value,address

              from v$sqlarea    where disk\_reads > 0 and executions >0

              order by disk\_reads desc) where rownum <= 10;

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Physical Reads** | **Executions** | **Reads per Exec** | **%Total** | **CPU Time (s)** | **Elapsed Time (s)** | **SQL Id** | **SQL Module** | **SQL Text** |
| 66,286 | 1 | 66,286.00 | 20.54 | 57.31 | 93.50 | [**d8z0u8hgj8xdy**](http://blog.itpub.net/26736162/viewspace-1869086/#d8z0u8hgj8xdy) | cuidmain@HPGICCI1 (TNS V1-V3) | insert into CUID select CUID\_... |
| 50,646 | 1 | 50,646.00 | 15.70 | 3.54 | 9.70 | [**0yv9t4qb1zb2b**](http://blog.itpub.net/26736162/viewspace-1869086/#0yv9t4qb1zb2b) | cuidmain@HPGICCI1 (TNS V1-V3) | select CUID\_CUST\_NO , CUID\_ID\_... |
| 24,507 | 1 | 24,507.00 | 7.59 | 42.01 | 58.04 | [**569r5k05drsj7**](http://blog.itpub.net/26736162/viewspace-1869086/#569r5k05drsj7) | cumimain@HPGICCI1 (TNS V1-V3) | insert into CUMI select CUSV\_... |
| 21,893 | 1 | 21,893.00 | 6.78 | 42.43 | 50.93 | [**ackxqhnktxnbc**](http://blog.itpub.net/26736162/viewspace-1869086/#ackxqhnktxnbc) | cusmmain@HPGICCI1 (TNS V1-V3) | insert into CUSM select CUSM\_... |
| 19,761 | 1 | 19,761.00 | 6.12 | 2.14 | 6.04 | [**a7nh7j8zmfrzw**](http://blog.itpub.net/26736162/viewspace-1869086/#a7nh7j8zmfrzw) | cumimain@HPGICCI1 (TNS V1-V3) | select CUSV\_CUST\_NO from CUMI... |
| 19,554 | 1 | 19,554.00 | 6.06 | 1.27 | 3.83 | [**38gak8u2qm11w**](http://blog.itpub.net/26736162/viewspace-1869086/#38gak8u2qm11w) | SQL\*Plus | select count(\*) from CUSVAA\_T... |
| 6,342 | 1 | 6,342.00 | 1.97 | 3.13 | 35.00 | [**6z06gcfw39pkd**](http://blog.itpub.net/26736162/viewspace-1869086/#6z06gcfw39pkd) | SQL\*Plus | SELECT F.TABLESPACE\_NAME, TO\_... |
| 4,385 | 1 | 4,385.00 | 1.36 | 1.59 | 2.43 | [**cp5duhcsj72q0**](http://blog.itpub.net/26736162/viewspace-1869086/#cp5duhcsj72q0) | cusmmain@HPGICCI1 (TNS V1-V3) | select CUSM\_CUST\_ACCT\_NO from... |
| 63 | 5 | 12.60 | 0.02 | 11.17 | 14.91 | [**djs2w2f17nw2z**](http://blog.itpub.net/26736162/viewspace-1869086/#djs2w2f17nw2z) |  | DECLARE job BINARY\_INTEGER := ... |
| 35 | 1 | 35.00 | 0.01 | 0.08 | 0.67 | [**1uk5m5qbzj1vt**](http://blog.itpub.net/26736162/viewspace-1869086/#1uk5m5qbzj1vt) | SQL\*Plus | BEGIN dbms\_workload\_repository... |

[**Back to SQL Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#22)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### SQL ordered by Executions

* **Total Executions: 1,675,112**
* **Captured SQL account for 99.8% of Total**

这部分告诉我们在这段时间中执行次数最多的SQL语句。为了隔离某些频繁执行的查询，以观察是否有某些更改逻辑的方法以避免必须如此频繁的执行这些查询，这可能是很有用的。或许一个查询正在一个循环的内部执行，而且它可能在循环的外部执行一次，可以设计简单的算法更改以减少必须执行这个查询的次数。即使它运行的飞快，任何被执行几百万次的操作都将开始耗尽大量的时间。

select \* from

              (select substr(sql\_text,1,40) sql,      executions,

rows\_processed, rows\_processed/executions "Rows/Exec",

              hash\_value,address

              from v$sqlarea    where executions > 0

              order by executions desc) where rownum <= 10 ;

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Executions** | **Rows Processed** | **Rows per Exec** | **CPU per Exec (s)** | **Elap per Exec (s)** | **SQL Id** | **SQL Module** | **SQL Text** |
| 172,983 | 172,329 | 1.00 | 0.00 | 0.00 | [**5c4qu2zmj3gux**](http://blog.itpub.net/26736162/viewspace-1869086/#5c4qu2zmj3gux) | load\_fnsact@HPGICCI1 (TNS V1-V3) | select \* from ICCIPRODCODE wh... |
| 172,983 | 172,329 | 1.00 | 0.00 | 0.00 | [**7wwv1ybs9zguz**](http://blog.itpub.net/26736162/viewspace-1869086/#7wwv1ybs9zguz) | load\_fnsact@HPGICCI1 (TNS V1-V3) | update ICCIFNSACT set BORM\_AD... |
| 172,337 | 172,337 | 1.00 | 0.00 | 0.00 | [**gmn2w09rdxn14**](http://blog.itpub.net/26736162/viewspace-1869086/#gmn2w09rdxn14) | load\_oldnewact@HPGICCI1 (TNS V1-V3) | insert into OLDNEWACT values ... |
| 172,329 | 172,329 | 1.00 | 0.00 | 0.00 | [**1dm3bq36vu3g8**](http://blog.itpub.net/26736162/viewspace-1869086/#1dm3bq36vu3g8) | load\_fnsact@HPGICCI1 (TNS V1-V3) | insert into iccifnsact values... |
| 172,329 | 172,329 | 1.00 | 0.00 | 0.00 | [**38apjgr0p55ns**](http://blog.itpub.net/26736162/viewspace-1869086/#38apjgr0p55ns) | load\_fnsact@HPGICCI1 (TNS V1-V3) | update ICCICCS set CCSMAXOVER... |
| 172,329 | 6,286 | 0.04 | 0.00 | 0.00 | [**4vja2k2gdtyup**](http://blog.itpub.net/26736162/viewspace-1869086/#4vja2k2gdtyup) | load\_fnsact@HPGICCI1 (TNS V1-V3) | insert into ICCICCS values (:... |
| 166,069 | 166,069 | 1.00 | 0.00 | 0.00 | [**7gtztzv329wg0**](http://blog.itpub.net/26736162/viewspace-1869086/#7gtztzv329wg0) |  | select c.name, u.name from co... |
| 166,052 | 166,052 | 1.00 | 0.00 | 0.00 | [**7v9dyf5r424yh**](http://blog.itpub.net/26736162/viewspace-1869086/#7v9dyf5r424yh) | icci\_migact@HPGICCI1 (TNS V1-V3) | select NEWACTNO into :b0 from... |
| 166,051 | 166,051 | 1.00 | 0.00 | 0.00 | [**chjmy0dxf9mbj**](http://blog.itpub.net/26736162/viewspace-1869086/#chjmy0dxf9mbj) | icci\_migact@HPGICCI1 (TNS V1-V3) | insert into ICCICCS values (:... |
| 51,740 | 51,740 | 1.00 | 0.00 | 0.00 | [**bu8tnqr3xv25q**](http://blog.itpub.net/26736162/viewspace-1869086/#bu8tnqr3xv25q) | load\_fnsact@HPGICCI1 (TNS V1-V3) | select count(\*) into :b0 fro... |
| 51,633 | 51,633 | 1.00 | 0.00 | 0.00 | [**49ms69srnaxzj**](http://blog.itpub.net/26736162/viewspace-1869086/#49ms69srnaxzj) | load\_fnsact@HPGICCI1 (TNS V1-V3) | insert into ICCIRPYV values (... |

[**Back to SQL Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#22)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### SQL ordered by Parse Calls

* **Total Parse Calls: 182,780**
* **Captured SQL account for 99.0% of Total**

在这一部分，主要显示PARSE与EXECUTIONS的对比情况。如果PARSE/EXECUTIONS>1，往往说明这个语句可能存在问题：没有使用绑定变量，共享池设置太小，cursor\_sharing被设置为exact，没有设置session\_cached\_cursors等等问题。

select \* from

              (select substr(sql\_text,1,40) sql,      parse\_calls,

executions, hash\_value,address

              from v$sqlarea    where parse\_calls > 0

              order by parse\_calls desc) where rownum <= 10 ;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Parse Calls** | **Executions** | **% Total Parses** | **SQL Id** | **SQL Module** | **SQL Text** |
| 166,069 | 166,069 | 90.86 | [**7gtztzv329wg0**](http://blog.itpub.net/26736162/viewspace-1869086/#7gtztzv329wg0) |  | select c.name, u.name from co... |
| 6,304 | 6,304 | 3.45 | [**2ym6hhaq30r73**](http://blog.itpub.net/26736162/viewspace-1869086/#2ym6hhaq30r73) |  | select type#, blocks, extents,... |
| 2,437 | 2,438 | 1.33 | [**bsa0wjtftg3uw**](http://blog.itpub.net/26736162/viewspace-1869086/#bsa0wjtftg3uw) |  | select file# from file$ where ... |
| 1,568 | 1,568 | 0.86 | [**9qgtwh66xg6nz**](http://blog.itpub.net/26736162/viewspace-1869086/#9qgtwh66xg6nz) |  | update seg$ set type#=:4, bloc... |
| 1,554 | 1,554 | 0.85 | [**aq4js2gkfjru8**](http://blog.itpub.net/26736162/viewspace-1869086/#aq4js2gkfjru8) |  | update tsq$ set blocks=:3, max... |
| 444 | 444 | 0.24 | [**104pd9mm3fh9p**](http://blog.itpub.net/26736162/viewspace-1869086/#104pd9mm3fh9p) |  | select blocks, maxblocks, gran... |
| 421 | 421 | 0.23 | [**350f5yrnnmshs**](http://blog.itpub.net/26736162/viewspace-1869086/#350f5yrnnmshs) |  | lock table sys.mon\_mods$ in ex... |
| 421 | 421 | 0.23 | [**g00cj285jmgsw**](http://blog.itpub.net/26736162/viewspace-1869086/#g00cj285jmgsw) |  | update sys.mon\_mods$ set inser... |
| 86 | 86 | 0.05 | [**3m8smr0v7v1m6**](http://blog.itpub.net/26736162/viewspace-1869086/#3m8smr0v7v1m6) |  | INSERT INTO sys.wri$\_adv\_messa... |
| 81 | 81 | 0.04 | [**f80h0xb1qvbsk**](http://blog.itpub.net/26736162/viewspace-1869086/#f80h0xb1qvbsk) |  | SELECT sys.wri$\_adv\_seq\_msggro... |

[**Back to SQL Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#22)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### SQL ordered by Sharable Memory

**No data exists for this section of the report.**

在这一部分，主要是针对shared memory占用的情况进行排序。

select \* from

              (select substr(sql\_text,1,40) sql,      sharable\_mem,

executions, hash\_value,address

              from v$sqlarea    where sharable\_mem > 1048576

              order by sharable\_mem desc)

              where rownum <= 10;

[**Back to SQL Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#22)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

**Running Time top 10 sql**

select \* from

              (select t.sql\_fulltext,

              (t.last\_active\_time-to\_date(t.first\_load\_time,'yyyy-mm-dd hh24:mi:ss'))\*24\*60,

              disk\_reads,buffer\_gets,rows\_processed,

              t.last\_active\_time,t.last\_load\_time,t.first\_load\_time

              from v$sqlarea t order by t.first\_load\_time desc)

              where rownum < 10;

### SQL ordered by Version Count

**No data exists for this section of the report.**

在这一部分，主要是针对SQL语句的多版本进行排序。相同的SQL文本，但是不同属性，比如对象owner不同，会话优化模式不同、类型不同、长度不同和绑定变量不同等等的语句，他们是不能共享的，所以再缓存中会存在多个不同的版本。这当然就造成了资源上的更多的消耗。

[**Back to SQL Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#22)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### SQL ordered by Cluster Wait Time

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Cluster Wait Time (s)** | **CWT % of Elapsd Time** | **Elapsed Time(s)** | **CPU Time(s)** | **Executions** | **SQL Id** | **SQL Module** | **SQL Text** |
| 10.96 | 11.72 | 93.50 | 57.31 | 1 | [**d8z0u8hgj8xdy**](http://blog.itpub.net/26736162/viewspace-1869086/#d8z0u8hgj8xdy) | cuidmain@HPGICCI1 (TNS V1-V3) | insert into CUID select CUID\_... |
| 4.21 | 7.25 | 58.04 | 42.01 | 1 | [**569r5k05drsj7**](http://blog.itpub.net/26736162/viewspace-1869086/#569r5k05drsj7) | cumimain@HPGICCI1 (TNS V1-V3) | insert into CUMI select CUSV\_... |
| 3.62 | 7.12 | 50.93 | 42.43 | 1 | [**ackxqhnktxnbc**](http://blog.itpub.net/26736162/viewspace-1869086/#ackxqhnktxnbc) | cusmmain@HPGICCI1 (TNS V1-V3) | insert into CUSM select CUSM\_... |
| 2.39 | 6.35 | 37.60 | 35.84 | 166,069 | [**7gtztzv329wg0**](http://blog.itpub.net/26736162/viewspace-1869086/#7gtztzv329wg0) |  | select c.name, u.name from co... |
| 2.38 | 3.12 | 76.41 | 74.57 | 172,329 | [**4vja2k2gdtyup**](http://blog.itpub.net/26736162/viewspace-1869086/#4vja2k2gdtyup) | load\_fnsact@HPGICCI1 (TNS V1-V3) | insert into ICCICCS values (:... |
| 1.64 | 16.91 | 9.70 | 3.54 | 1 | [**0yv9t4qb1zb2b**](http://blog.itpub.net/26736162/viewspace-1869086/#0yv9t4qb1zb2b) | cuidmain@HPGICCI1 (TNS V1-V3) | select CUID\_CUST\_NO , CUID\_ID\_... |
| 1.06 | 3.02 | 35.00 | 3.13 | 1 | [**6z06gcfw39pkd**](http://blog.itpub.net/26736162/viewspace-1869086/#6z06gcfw39pkd) | SQL\*Plus | SELECT F.TABLESPACE\_NAME, TO\_... |
| 0.83 | 13.76 | 6.04 | 2.14 | 1 | [**a7nh7j8zmfrzw**](http://blog.itpub.net/26736162/viewspace-1869086/#a7nh7j8zmfrzw) | cumimain@HPGICCI1 (TNS V1-V3) | select CUSV\_CUST\_NO from CUMI... |
| 0.66 | 87.90 | 0.75 | 0.42 | 444 | [**104pd9mm3fh9p**](http://blog.itpub.net/26736162/viewspace-1869086/#104pd9mm3fh9p) |  | select blocks, maxblocks, gran... |
| 0.50 | 13.01 | 3.83 | 1.27 | 1 | [**38gak8u2qm11w**](http://blog.itpub.net/26736162/viewspace-1869086/#38gak8u2qm11w) | SQL\*Plus | select count(\*) from CUSVAA\_T... |
| 0.50 | 51.75 | 0.96 | 0.79 | 1,554 | [**aq4js2gkfjru8**](http://blog.itpub.net/26736162/viewspace-1869086/#aq4js2gkfjru8) |  | update tsq$ set blocks=:3, max... |
| 0.33 | 91.11 | 0.36 | 0.33 | 187 | [**04xtrk7uyhknh**](http://blog.itpub.net/26736162/viewspace-1869086/#04xtrk7uyhknh) |  | select obj#, type#, ctime, mti... |
| 0.33 | 2.47 | 13.29 | 12.97 | 172,337 | [**gmn2w09rdxn14**](http://blog.itpub.net/26736162/viewspace-1869086/#gmn2w09rdxn14) | load\_oldnewact@HPGICCI1 (TNS V1-V3) | insert into OLDNEWACT values ... |
| 0.29 | 1.26 | 22.94 | 22.85 | 172,329 | [**1dm3bq36vu3g8**](http://blog.itpub.net/26736162/viewspace-1869086/#1dm3bq36vu3g8) | load\_fnsact@HPGICCI1 (TNS V1-V3) | insert into iccifnsact values... |
| 0.25 | 10.14 | 2.43 | 1.59 | 1 | [**cp5duhcsj72q0**](http://blog.itpub.net/26736162/viewspace-1869086/#cp5duhcsj72q0) | cusmmain@HPGICCI1 (TNS V1-V3) | select CUSM\_CUST\_ACCT\_NO from... |
| 0.21 | 27.92 | 0.74 | 0.74 | 1,568 | [**9qgtwh66xg6nz**](http://blog.itpub.net/26736162/viewspace-1869086/#9qgtwh66xg6nz) |  | update seg$ set type#=:4, bloc... |
| 0.20 | 3.49 | 5.83 | 5.75 | 51,633 | [**49ms69srnaxzj**](http://blog.itpub.net/26736162/viewspace-1869086/#49ms69srnaxzj) | load\_fnsact@HPGICCI1 (TNS V1-V3) | insert into ICCIRPYV values (... |
| 0.17 | 1.39 | 12.55 | 12.52 | 166,051 | [**chjmy0dxf9mbj**](http://blog.itpub.net/26736162/viewspace-1869086/#chjmy0dxf9mbj) | icci\_migact@HPGICCI1 (TNS V1-V3) | insert into ICCICCS values (:... |
| 0.16 | 57.64 | 0.28 | 0.24 | 39 | [**cn1gtsav2d5jh**](http://blog.itpub.net/26736162/viewspace-1869086/#cn1gtsav2d5jh) | cusvaamain@HPGICCI1 (TNS V1-V3) | BEGIN BEGIN IF (xdb.DBMS... |
| 0.14 | 74.58 | 0.19 | 0.14 | 121 | [**5ngzsfstg8tmy**](http://blog.itpub.net/26736162/viewspace-1869086/#5ngzsfstg8tmy) |  | select o.owner#, o.name, o.nam... |
| 0.11 | 64.72 | 0.18 | 0.15 | 80 | [**78m9ryygp65v5**](http://blog.itpub.net/26736162/viewspace-1869086/#78m9ryygp65v5) | cusvaamain@HPGICCI1 (TNS V1-V3) | SELECT /\*+ ALL\_ROWS \*/ COUNT(\*... |
| 0.11 | 94.54 | 0.12 | 0.01 | 17 | [**bwt0pmxhv7qk7**](http://blog.itpub.net/26736162/viewspace-1869086/#bwt0pmxhv7qk7) |  | delete from con$ where owner#=... |
| 0.11 | 80.26 | 0.14 | 0.14 | 327 | [**53saa2zkr6wc3**](http://blog.itpub.net/26736162/viewspace-1869086/#53saa2zkr6wc3) |  | select intcol#, nvl(pos#, 0), ... |
| 0.08 | 19.20 | 0.42 | 0.24 | 1 | [**d92h3rjp0y217**](http://blog.itpub.net/26736162/viewspace-1869086/#d92h3rjp0y217) |  | begin prvt\_hdm.auto\_execute( :... |
| 0.07 | 54.97 | 0.13 | 0.13 | 83 | [**7ng34ruy5awxq**](http://blog.itpub.net/26736162/viewspace-1869086/#7ng34ruy5awxq) |  | select i.obj#, i.ts#, i.file#,... |
| 0.06 | 5.22 | 1.13 | 0.72 | 77 | [**0hhmdwwgxbw0r**](http://blog.itpub.net/26736162/viewspace-1869086/#0hhmdwwgxbw0r) |  | select obj#, type#, flags, ... |
| 0.06 | 86.50 | 0.06 | 0.06 | 45 | [**a2any035u1qz1**](http://blog.itpub.net/26736162/viewspace-1869086/#a2any035u1qz1) |  | select owner#, name from con$... |
| 0.06 | 8.19 | 0.67 | 0.08 | 1 | [**1uk5m5qbzj1vt**](http://blog.itpub.net/26736162/viewspace-1869086/#1uk5m5qbzj1vt) | SQL\*Plus | BEGIN dbms\_workload\_repository... |
| 0.04 | 75.69 | 0.06 | 0.06 | 87 | [**6769wyy3yf66f**](http://blog.itpub.net/26736162/viewspace-1869086/#6769wyy3yf66f) |  | select pos#, intcol#, col#, sp... |
| 0.04 | 48.05 | 0.09 | 0.07 | 7 | [**0pvtkmrrq8usg**](http://blog.itpub.net/26736162/viewspace-1869086/#0pvtkmrrq8usg) |  | select file#, block# from seg... |
| 0.04 | 8.84 | 0.40 | 0.40 | 6,304 | [**2ym6hhaq30r73**](http://blog.itpub.net/26736162/viewspace-1869086/#2ym6hhaq30r73) |  | select type#, blocks, extents,... |
| 0.03 | 28.15 | 0.12 | 0.12 | 49 | [**b52m6vduutr8j**](http://blog.itpub.net/26736162/viewspace-1869086/#b52m6vduutr8j) |  | delete from RecycleBin$ ... |
| 0.03 | 66.23 | 0.05 | 0.05 | 85 | [**1gu8t96d0bdmu**](http://blog.itpub.net/26736162/viewspace-1869086/#1gu8t96d0bdmu) |  | select t.ts#, t.file#, t.block... |
| 0.03 | 67.03 | 0.05 | 0.05 | 38 | [**btzq46kta67dz**](http://blog.itpub.net/26736162/viewspace-1869086/#btzq46kta67dz) | DBMS\_SCHEDULER | update obj$ set obj#=:6, type#... |
| 0.02 | 66.73 | 0.04 | 0.04 | 86 | [**3m8smr0v7v1m6**](http://blog.itpub.net/26736162/viewspace-1869086/#3m8smr0v7v1m6) |  | INSERT INTO sys.wri$\_adv\_messa... |
| 0.02 | 26.94 | 0.09 | 0.09 | 38 | [**0k8h617b8guhf**](http://blog.itpub.net/26736162/viewspace-1869086/#0k8h617b8guhf) |  | delete from RecycleBin$ ... |
| 0.02 | 76.76 | 0.03 | 0.03 | 51 | [**9vtm7gy4fr2ny**](http://blog.itpub.net/26736162/viewspace-1869086/#9vtm7gy4fr2ny) |  | select con# from con$ where ow... |
| 0.02 | 51.91 | 0.05 | 0.05 | 84 | [**83taa7kaw59c1**](http://blog.itpub.net/26736162/viewspace-1869086/#83taa7kaw59c1) |  | select name, intcol#, segcol#,... |
| 0.02 | 0.15 | 14.91 | 11.17 | 5 | [**djs2w2f17nw2z**](http://blog.itpub.net/26736162/viewspace-1869086/#djs2w2f17nw2z) |  | DECLARE job BINARY\_INTEGER := ... |
| 0.02 | 2.12 | 1.00 | 0.99 | 8,784 | [**501v412s13r4m**](http://blog.itpub.net/26736162/viewspace-1869086/#501v412s13r4m) | load\_fnsact@HPGICCI1 (TNS V1-V3) | update ICCIFNSACT set BORM\_FA... |
| 0.02 | 53.82 | 0.03 | 0.03 | 39 | [**bdv0rkkssq2jm**](http://blog.itpub.net/26736162/viewspace-1869086/#bdv0rkkssq2jm) | cusvaamain@HPGICCI1 (TNS V1-V3) | SELECT count(\*) FROM user\_poli... |
| 0.01 | 0.10 | 14.34 | 14.28 | 172,983 | [**7wwv1ybs9zguz**](http://blog.itpub.net/26736162/viewspace-1869086/#7wwv1ybs9zguz) | load\_fnsact@HPGICCI1 (TNS V1-V3) | update ICCIFNSACT set BORM\_AD... |
| 0.01 | 8.29 | 0.16 | 0.13 | 421 | [**g00cj285jmgsw**](http://blog.itpub.net/26736162/viewspace-1869086/#g00cj285jmgsw) |  | update sys.mon\_mods$ set inser... |
| 0.01 | 1.65 | 0.56 | 0.54 | 2 | [**84qubbrsr0kfn**](http://blog.itpub.net/26736162/viewspace-1869086/#84qubbrsr0kfn) |  | insert into wrh$\_latch (snap... |
| 0.01 | 22.33 | 0.04 | 0.02 | 26 | [**44au3v5mzpc1c**](http://blog.itpub.net/26736162/viewspace-1869086/#44au3v5mzpc1c) | load\_curmmast@HPGICCI1 (TNS V1-V3) | insert into ICCICURMMAST valu... |
| 0.01 | 0.08 | 7.71 | 7.70 | 172,983 | [**5c4qu2zmj3gux**](http://blog.itpub.net/26736162/viewspace-1869086/#5c4qu2zmj3gux) | load\_fnsact@HPGICCI1 (TNS V1-V3) | select \* from ICCIPRODCODE wh... |

[**Back to SQL Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#22)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

对于出现在上面的可疑的sql语句，我们可以查看语句相关的执行计划，然后分析相关索引等是否合理。

通过语句查看执行计划的方法:

SELECT id,parent\_id,LPAD(' ',4\*(LEVEL-1))||operation||' '||options||' '||object\_name "Execution plan" ,cost,cardinality,bytes

FROM (

SELECT p.\* FROM v$sql\_plan p,v$sql s WHERE p.address = s.ADDRESS

AND p.hash\_value = s.HASH\_VALUE

and p.hash\_value = '&hash\_value'

)

CONNECT BY PRIOR id = parent\_id

START WITH id = 0;

    查看，分析，优化索引等在这里就不再一一描述了。

### Complete List of SQL Text

|  |  |
| --- | --- |
| **SQL Id** | **SQL Text** |
| 04xtrk7uyhknh | select obj#, type#, ctime, mtime, stime, status, dataobj#, flags, oid$, spare1, spare2 from obj$ where owner#=:1 and name=:2 and namespace=:3 and remoteowner is null and linkname is null and subname is null |
| 0hhmdwwgxbw0r | select obj#, type#, flags, related, bo, purgeobj, con# from RecycleBin$ where ts#=:1 and to\_number(bitand(flags, 16)) = 16 order by dropscn |
| 0k8h617b8guhf | delete from RecycleBin$ where purgeobj=:1 |
| 0pvtkmrrq8usg | select file#, block# from seg$ where type# = 3 and ts# = :1 |
| 0v9t4qb1zb2b | select CUID\_CUST\_NO , CUID\_ID\_TYPE , CUID\_ID\_RECNO from CUID\_TMP where CHGFLAG='D' |
| 104pd9mm3fh9p | select blocks, maxblocks, grantor#, priv1, priv2, priv3 from tsq$ where ts#=:1 and user#=:2 |
| 1crajpb7j5tyz | INSERT INTO STATS$SGA\_TARGET\_ADVICE ( SNAP\_ID , DBID , INSTANCE\_NUMBER , SGA\_SIZE , SGA\_SIZE\_FACTOR , ESTD\_DB\_TIME , ESTD\_DB\_TIME\_FACTOR , ESTD\_PHYSICAL\_READS ) SELECT :B3 , :B2 , :B1 , SGA\_SIZE , SGA\_SIZE\_FACTOR , ESTD\_DB\_TIME , ESTD\_DB\_TIME\_FACTOR , ESTD\_PHYSICAL\_READS FROM V$SGA\_TARGET\_ADVICE |
| 1dm3bq36vu3g8 | insert into iccifnsact values (:b0, :b1, :b2, null , null , :b3, :b4, GREATEST(:b5, :b6), null , :b7, :b8, null , :b9, :b10, :b6, null , null , null , null , null , :b12, null , null , null , :b13, :b14, null , null , :b15, :b16, :b17) |
| 1gu8t96d0bdmu | select t.ts#, t.file#, t.block#, nvl(t.bobj#, 0), nvl(t.tab#, 0), t.intcols, nvl(t.clucols, 0), t.audit$, t.flags, t.pctfree$, t.pctused$, t.initrans, t.maxtrans, t.rowcnt, t.blkcnt, t.empcnt, t.avgspc, t.chncnt, t.avgrln, t.analyzetime, t.samplesize, t.cols, t.property, nvl(t.degree, 1), nvl(t.instances, 1), t.avgspc\_flb, t.flbcnt, t.kernelcols, nvl(t.trigflag, 0), nvl(t.spare1, 0), nvl(t.spare2, 0), t.spare4, t.spare6, ts.cachedblk, ts.cachehit, ts.logicalread from tab$ t, tab\_stats$ ts where t.obj#= :1 and t.obj# = ts.obj# (+) |
| 1uk5m5qbzj1vt | BEGIN dbms\_workload\_repository.create\_snapshot; END; |
| 2ym6hhaq30r73 | select type#, blocks, extents, minexts, maxexts, extsize, extpct, user#, iniexts, NVL(lists, 65535), NVL(groups, 65535), cachehint, hwmincr, NVL(spare1, 0), NVL(scanhint, 0) from seg$ where ts#=:1 and file#=:2 and block#=:3 |
| 350f5yrnnmshs | lock table sys.mon\_mods$ in exclusive mode nowait |
| 38apjgr0p55ns | update ICCICCS set CCSMAXOVERDUE=GREATEST(:b0, CCSMAXOVERDUE) where FNSACTNO=:b1 |
| 38gak8u2qm11w | select count(\*) from CUSVAA\_TMP |
| 3m8smr0v7v1m6 | INSERT INTO sys.wri$\_adv\_message\_groups (task\_id, id, seq, message#, fac, hdr, lm, nl, p1, p2, p3, p4, p5) VALUES (:1, :2, :3, :4, :5, :6, :7, :8, :9, :10, :11, :12, :13) |
| 44au3v5mzpc1c | insert into ICCICURMMAST values (:b0, :b1, :b2) |
| 49ms69srnaxzj | insert into ICCIRPYV values (:b0, :b1, :b2, :b3, :b4, :b5, :b6, :b7, :b8, :b9, :b10, :b11, :b12, :b13, :b14, :b15, :b16, :b17, :b18, :b19, :b20, :b21, :b22, :b23, :b24, :b25, :b26, :b27, :b28, :b29, :b30, :b31, :b32, :b33, :b34, :b35, :b36, :b37, :b38, :b39, :b40, :b41, :b42, :b43, :b44, :b45, :b46, :b47, :b48, :b49, :b50, :b51) |
| 4vja2k2gdtyup | insert into ICCICCS values (:b0, '////////////////////////', 0, 0, 0, 0, 0, ' ', 0, 0, 0, ' ', '0', null ) |
| 501v412s13r4m | update ICCIFNSACT set BORM\_FACILITY\_NO=:b0 where BORM\_MEMB\_CUST\_AC=:b1 |
| 53saa2zkr6wc3 | select intcol#, nvl(pos#, 0), col#, nvl(spare1, 0) from ccol$ where con#=:1 |
| 569r5k05drsj7 | insert into CUMI select CUSV\_CUST\_NO , CUSV\_EDUCATION\_CODE , CHGDATE from CUMI\_TMP where CHGFLAG<>'D' |
| 5c4qu2zmj3gux | select \* from ICCIPRODCODE where PRODCODE=to\_char(:b0) |
| 5ngzsfstg8tmy | select o.owner#, o.name, o.namespace, o.remoteowner, o.linkname, o.subname, o.dataobj#, o.flags from obj$ o where o.obj#=:1 |
| 6769wyy3yf66f | select pos#, intcol#, col#, spare1, bo#, spare2 from icol$ where obj#=:1 |
| 6z06gcfw39pkd | SELECT F.TABLESPACE\_NAME, TO\_CHAR ((T.TOTAL\_SPACE - F.FREE\_SPACE), '999, 999') "USED (MB)", TO\_CHAR (F.FREE\_SPACE, '999, 999') "FREE (MB)", TO\_CHAR (T.TOTAL\_SPACE, '999, 999') "TOTAL (MB)", TO\_CHAR ((ROUND ((F.FREE\_SPACE/T.TOTAL\_SPACE)\*100)), '999')||' %' PER\_FREE FROM ( SELECT TABLESPACE\_NAME, ROUND (SUM (BLOCKS\*(SELECT VALUE/1024 FROM V$PARAMETER WHERE NAME = 'db\_block\_size')/1024) ) FREE\_SPACE FROM DBA\_FREE\_SPACE GROUP BY TABLESPACE\_NAME ) F, ( SELECT TABLESPACE\_NAME, ROUND (SUM (BYTES/1048576)) TOTAL\_SPACE FROM DBA\_DATA\_FILES GROUP BY TABLESPACE\_NAME ) T WHERE F.TABLESPACE\_NAME = T.TABLESPACE\_NAME |
| 78m9ryygp65v5 | SELECT /\*+ ALL\_ROWS \*/ COUNT(\*) FROM ALL\_POLICIES V WHERE V.OBJECT\_OWNER = :B3 AND V.OBJECT\_NAME = :B2 AND (POLICY\_NAME LIKE '%xdbrls%' OR POLICY\_NAME LIKE '%$xd\_%') AND V.FUNCTION = :B1 |
| 7gtztzv329wg0 | select c.name, u.name from con$ c, cdef$ cd, user$ u where c.con# = cd.con# and cd.enabled = :1 and c.owner# = u.user# |
| 7ng34ruy5awxq | select i.obj#, i.ts#, i.file#, i.block#, i.intcols, i.type#, i.flags, i.property, i.pctfree$, i.initrans, i.maxtrans, i.blevel, i.leafcnt, i.distkey, i.lblkkey, i.dblkkey, i.clufac, i.cols, i.analyzetime, i.samplesize, i.dataobj#, nvl(i.degree, 1), nvl(i.instances, 1), i.rowcnt, mod(i.pctthres$, 256), i.indmethod#, i.trunccnt, nvl(c.unicols, 0), nvl(c.deferrable#+c.valid#, 0), nvl(i.spare1, i.intcols), i.spare4, i.spare2, i.spare6, decode(i.pctthres$, null, null, mod(trunc(i.pctthres$/256), 256)), ist.cachedblk, ist.cachehit, ist.logicalread from ind$ i, ind\_stats$ ist, (select enabled, min(cols) unicols, min(to\_number(bitand(defer, 1))) deferrable#, min(to\_number(bitand(defer, 4))) valid# from cdef$ where obj#=:1 and enabled > 1 group by enabled) c where i.obj#=c.enabled(+) and i.obj# = ist.obj#(+) and i.bo#=:1 order by i.obj# |
| 7v9dyf5r424yh | select NEWACTNO into :b0 from OLDNEWACT where OLDACTNO=:b1 |
| 7wwv1ybs9zguz | update ICCIFNSACT set BORM\_ADV\_DATE=:b0, BOIS\_MATURITY\_DATE=:b1, BOIS\_UNPD\_BAL=:b2, BOIS\_UNPD\_INT=:b3, BOIS\_BAL\_FINE=:b4, BOIS\_INT\_FINE=:b5, BOIS\_FINE\_FINE=:b6, BORM\_LOAN\_TRM=:b7, BORM\_FIVE\_STAT=:b8, BOIS\_ARREARS\_CTR=:b9, BOIS\_ARREARS\_SUM=:b10 where BORM\_MEMB\_CUST\_AC=:b11 |
| 83taa7kaw59c1 | select name, intcol#, segcol#, type#, length, nvl(precision#, 0), decode(type#, 2, nvl(scale, -127/\*MAXSB1MINAL\*/), 178, scale, 179, scale, 180, scale, 181, scale, 182, scale, 183, scale, 231, scale, 0), null$, fixedstorage, nvl(deflength, 0), default$, rowid, col#, property, nvl(charsetid, 0), nvl(charsetform, 0), spare1, spare2, nvl(spare3, 0) from col$ where obj#=:1 order by intcol# |
| 4qubbrsr0kfn | insert into wrh$\_latch (snap\_id, dbid, instance\_number, latch\_hash, level#, gets, misses, sleeps, immediate\_gets, immediate\_misses, spin\_gets, sleep1, sleep2, sleep3, sleep4, wait\_time) select :snap\_id, :dbid, :instance\_number, hash, level#, gets, misses, sleeps, immediate\_gets, immediate\_misses, spin\_gets, sleep1, sleep2, sleep3, sleep4, wait\_time from v$latch order by hash |
| 9qgtwh66xg6nz | update seg$ set type#=:4, blocks=:5, extents=:6, minexts=:7, maxexts=:8, extsize=:9, extpct=:10, user#=:11, iniexts=:12, lists=decode(:13, 65535, NULL, :13), groups=decode(:14, 65535, NULL, :14), cachehint=:15, hwmincr=:16, spare1=DECODE(:17, 0, NULL, :17), scanhint=:18 where ts#=:1 and file#=:2 and block#=:3 |
| 9vtm7gy4fr2ny | select con# from con$ where owner#=:1 and name=:2 |
| a2any035u1qz1 | select owner#, name from con$ where con#=:1 |
| a7nh7j8zmfrzw | select CUSV\_CUST\_NO from CUMI\_TMP where CHGFLAG='D' |
|  |  |

[**Back to SQL Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#22)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

## Instance Activity Statistics

* [**Instance Activity Stats**](http://blog.itpub.net/26736162/viewspace-1869086/#1200)
* [**Instance Activity Stats - Absolute Values**](http://blog.itpub.net/26736162/viewspace-1869086/#1300)
* [**Instance Activity Stats - Thread Activity**](http://blog.itpub.net/26736162/viewspace-1869086/#1400)

[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Instance Activity Stats

|  |  |  |  |
| --- | --- | --- | --- |
| **Statistic** | **Total** | **per Second** | **per Trans** |
| CPU used by this session | 23,388 | 4.95 | 4.18 |
| CPU used when call started | 21,816 | 4.61 | 3.90 |
| CR blocks created | 2,794 | 0.59 | 0.50 |
| Cached Commit SCN referenced | 237,936 | 50.33 | 42.50 |
| Commit SCN cached | 3 | 0.00 | 0.00 |
| DB time | 583,424 | 123.41 | 104.22 |
| DBWR checkpoint buffers written | 402,781 | 85.20 | 71.95 |
| DBWR checkpoints | 9 | 0.00 | 0.00 |
| DBWR fusion writes | 255 | 0.05 | 0.05 |
| DBWR object drop buffers written | 0 | 0.00 | 0.00 |
| DBWR thread checkpoint buffers written | 221,341 | 46.82 | 39.54 |
| DBWR transaction table writes | 130 | 0.03 | 0.02 |
| DBWR undo block writes | 219,272 | 46.38 | 39.17 |
| DFO trees parallelized | 16 | 0.00 | 0.00 |
| PX local messages recv'd | 40 | 0.01 | 0.01 |
| PX local messages sent | 40 | 0.01 | 0.01 |
| PX remote messages recv'd | 80 | 0.02 | 0.01 |
| PX remote messages sent | 80 | 0.02 | 0.01 |
| Parallel operations not downgraded | 16 | 0.00 | 0.00 |
| RowCR - row contention | 9 | 0.00 | 0.00 |
| RowCR attempts | 14 | 0.00 | 0.00 |
| RowCR hits | 5 | 0.00 | 0.00 |
| SMON posted for undo segment recovery | 0 | 0.00 | 0.00 |
| SMON posted for undo segment shrink | 9 | 0.00 | 0.00 |
| SQL\*Net roundtrips to/from client | 1,544,063 | 326.62 | 275.82 |
| active txn count during cleanout | 276,652 | 58.52 | 49.42 |
| application wait time | 1,620 | 0.34 | 0.29 |
| auto extends on undo tablespace | 0 | 0.00 | 0.00 |
| background checkpoints completed | 7 | 0.00 | 0.00 |
| background checkpoints started | 9 | 0.00 | 0.00 |
| background timeouts | 21,703 | 4.59 | 3.88 |
| branch node splits | 337 | 0.07 | 0.06 |
| buffer is not pinned count | 1,377,184 | 291.32 | 246.01 |
| buffer is pinned count | 20,996,139 | 4,441.37 | 3,750.65 |
| bytes received via SQL\*Net from client | 7,381,397,183 | 1,561,408.36 | 1,318,577.56 |
| bytes sent via SQL\*Net to client | 149,122,035 | 31,544.22 | 26,638.45 |
| calls to get snapshot scn: kcmgss | 1,696,712 | 358.91 | 303.09 |
| calls to kcmgas | 433,435 | 91.69 | 77.43 |
| calls to kcmgcs | 142,482 | 30.14 | 25.45 |
| change write time | 4,707 | 1.00 | 0.84 |
| cleanout - number of ktugct calls | 282,045 | 59.66 | 50.38 |
| cleanouts and rollbacks - consistent read gets | 55 | 0.01 | 0.01 |
| cleanouts only - consistent read gets | 2,406 | 0.51 | 0.43 |
| cluster key scan block gets | 21,886 | 4.63 | 3.91 |
| cluster key scans | 10,540 | 2.23 | 1.88 |
| cluster wait time | 2,855 | 0.60 | 0.51 |
| commit batch/immediate performed | 294 | 0.06 | 0.05 |
| commit batch/immediate requested | 294 | 0.06 | 0.05 |
| commit cleanout failures: block lost | 2,227 | 0.47 | 0.40 |
| commit cleanout failures: callback failure | 750 | 0.16 | 0.13 |
| commit cleanout failures: cannot pin | 4 | 0.00 | 0.00 |
| commit cleanouts | 427,610 | 90.45 | 76.39 |
| commit cleanouts successfully completed | 424,629 | 89.82 | 75.85 |
| commit immediate performed | 294 | 0.06 | 0.05 |
| commit immediate requested | 294 | 0.06 | 0.05 |
| commit txn count during cleanout | 111,557 | 23.60 | 19.93 |
| concurrency wait time | 515 | 0.11 | 0.09 |
| consistent changes | 1,716 | 0.36 | 0.31 |
| consistent gets | 5,037,471 | 1,065.59 | 899.87 |
| 由consistent gets，db block gets和physical reads这三个值，我们也可以计算得到buffer hit ratio，计算的公式如下： buffer hit ratio = 100\*(1-physical reads /(consistent gets+ db block gets))，例如在这里，我们可以计算得到：buffer hit ratio =100\*(1-26524/(16616758+2941398))= 99.86 | | | |
| consistent gets - examination | 2,902,016 | 613.87 | 518.40 |
| consistent gets direct | 0 | 0.00 | 0.00 |
| consistent gets from cache | 5,037,471 | 1,065.59 | 899.87 |
| current blocks converted for CR | 0 | 0.00 | 0.00 |
| cursor authentications | 434 | 0.09 | 0.08 |
| data blocks consistent reads - undo records applied | 1,519 | 0.32 | 0.27 |
| db block changes | 8,594,158 | 1,817.95 | 1,535.22 |
| db block gets | 11,611,321 | 2,456.18 | 2,074.19 |
| db block gets direct | 1,167,830 | 247.03 | 208.62 |
| db block gets from cache | 10,443,491 | 2,209.14 | 1,865.58 |
| deferred (CURRENT) block cleanout applications | 20,786 | 4.40 | 3.71 |
| dirty buffers inspected | 25,007 | 5.29 | 4.47 |
| 脏数据从LRU列表中老化，A value here indicates that the DBWR is not keeping up。如果这个值大于0，就需要考虑增加DBWRs。  dirty buffers inspected: This is the number of dirty (modified) data buffers that were aged out on the LRU list. You may benefit by adding more DBWRs.If it is greater than 0, consider increasing the database writes. | | | |
| drop segment calls in space pressure | 0 | 0.00 | 0.00 |
| enqueue conversions | 6,734 | 1.42 | 1.20 |
| enqueue releases | 595,149 | 125.89 | 106.31 |
| enqueue requests | 595,158 | 125.90 | 106.32 |
| enqueue timeouts | 9 | 0.00 | 0.00 |
| enqueue waits | 7,901 | 1.67 | 1.41 |
| exchange deadlocks | 1 | 0.00 | 0.00 |
| execute count | 1,675,112 | 354.34 | 299.23 |
| free buffer inspected | 536,832 | 113.56 | 95.90 |
| 这个值包含dirty，pinned，busy的buffer区域，如果free buffer inspected - dirty buffers inspected - buffer is pinned count的值还是比较大，表明不能被重用的内存块比较多，这将导致latch争用，需要增大buffer cache | | | |
| free buffer requested | 746,999 | 158.01 | 133.44 |
| gc CPU used by this session | 9,099 | 1.92 | 1.63 |
| gc cr block build time | 13 | 0.00 | 0.00 |
| gc cr block flush time | 143 | 0.03 | 0.03 |
| gc cr block receive time | 474 | 0.10 | 0.08 |
| gc cr block send time | 36 | 0.01 | 0.01 |
| gc cr blocks received | 4,142 | 0.88 | 0.74 |
| gc cr blocks served | 10,675 | 2.26 | 1.91 |
| gc current block flush time | 23 | 0.00 | 0.00 |
| gc current block pin time | 34 | 0.01 | 0.01 |
| gc current block receive time | 1,212 | 0.26 | 0.22 |
| gc current block send time | 52 | 0.01 | 0.01 |
| gc current blocks received | 15,502 | 3.28 | 2.77 |
| gc current blocks served | 17,534 | 3.71 | 3.13 |
| gc local grants | 405,329 | 85.74 | 72.41 |
| gc remote grants | 318,630 | 67.40 | 56.92 |
| gcs messages sent | 1,129,094 | 238.84 | 201.70 |
| ges messages sent | 90,695 | 19.18 | 16.20 |
| global enqueue get time | 1,707 | 0.36 | 0.30 |
| global enqueue gets async | 12,731 | 2.69 | 2.27 |
| global enqueue gets sync | 190,492 | 40.30 | 34.03 |
| global enqueue releases | 190,328 | 40.26 | 34.00 |
| global undo segment hints helped | 0 | 0.00 | 0.00 |
| global undo segment hints were stale | 0 | 0.00 | 0.00 |
| heap block compress | 108,758 | 23.01 | 19.43 |
| hot buffers moved to head of LRU | 18,652 | 3.95 | 3.33 |
| immediate (CR) block cleanout applications | 2,462 | 0.52 | 0.44 |
| immediate (CURRENT) block cleanout applications | 325,184 | 68.79 | 58.09 |
| index crx upgrade (positioned) | 4,663 | 0.99 | 0.83 |
| index fast full scans (full) | 13 | 0.00 | 0.00 |
| index fetch by key | 852,181 | 180.26 | 152.23 |
| index scans kdiixs1 | 339,583 | 71.83 | 60.66 |
| leaf node 90-10 splits | 34 | 0.01 | 0.01 |
| leaf node splits | 106,552 | 22.54 | 19.03 |
| lob reads | 11 | 0.00 | 0.00 |
| lob writes | 83 | 0.02 | 0.01 |
| lob writes unaligned | 83 | 0.02 | 0.01 |
| local undo segment hints helped | 0 | 0.00 | 0.00 |
| local undo segment hints were stale | 0 | 0.00 | 0.00 |
| logons cumulative | 61 | 0.01 | 0.01 |
| messages received | 20,040 | 4.24 | 3.58 |
| messages sent | 19,880 | 4.21 | 3.55 |
| no buffer to keep pinned count | 0 | 0.00 | 0.00 |
| no work - consistent read gets | 1,513,070 | 320.06 | 270.29 |
| opened cursors cumulative | 183,375 | 38.79 | 32.76 |
| parse count (failures) | 1 | 0.00 | 0.00 |
| parse count (hard) | 143 | 0.03 | 0.03 |
| parse count (total) | 182,780 | 38.66 | 32.65 |
| 通过parse count (hard)和parse count (total)，可以计算soft parse率为：  100-100\*(parse count (hard)/parse count (total)) =100-100\*(1-6090/191531)=96.82 | | | |
| parse time cpu | 27 | 0.01 | 0.00 |
| parse time elapsed | 338 | 0.07 | 0.06 |
| physical read IO requests | 82,815 | 17.52 | 14.79 |
| physical read bytes | 2,643,378,176 | 559,161.45 | 472,200.46 |
| physical read total IO requests | 98,871 | 20.91 | 17.66 |
| physical read total bytes | 2,905,491,456 | 614,607.04 | 519,023.13 |
| physical read total multi block requests | 24,089 | 5.10 | 4.30 |
| physical reads | 322,678 | 68.26 | 57.64 |
| physical reads cache | 213,728 | 45.21 | 38.18 |
| physical reads cache prefetch | 191,830 | 40.58 | 34.27 |
| physical reads direct | 108,950 | 23.05 | 19.46 |
| physical reads direct temporary tablespace | 108,812 | 23.02 | 19.44 |
| physical reads prefetch warmup | 0 | 0.00 | 0.00 |
| physical write IO requests | 223,456 | 47.27 | 39.92 |
| physical write bytes | 14,042,071,040 | 2,970,360.02 | 2,508,408.55 |
| physical write total IO requests | 133,835 | 28.31 | 23.91 |
| physical write total bytes | 23,114,268,672 | 4,889,428.30 | 4,129,022.63 |
| physical write total multi block requests | 116,135 | 24.57 | 20.75 |
| physical writes | 1,714,120 | 362.59 | 306.20 |
| physical writes direct | 1,276,780 | 270.08 | 228.08 |
| physical writes direct (lob) | 0 | 0.00 | 0.00 |
| physical writes direct temporary tablespace | 108,812 | 23.02 | 19.44 |
| physical writes from cache | 437,340 | 92.51 | 78.12 |
| physical writes non checkpoint | 1,673,703 | 354.04 | 298.98 |
| pinned buffers inspected | 10 | 0.00 | 0.00 |
| prefetch clients - default | 0 | 0.00 | 0.00 |
| prefetch warmup blocks aged out before use | 0 | 0.00 | 0.00 |
| prefetch warmup blocks flushed out before use | 0 | 0.00 | 0.00 |
| prefetched blocks aged out before use | 0 | 0.00 | 0.00 |
| process last non-idle time | 4,730 | 1.00 | 0.84 |
| queries parallelized | 16 | 0.00 | 0.00 |
| recursive calls | 1,654,650 | 350.01 | 295.58 |
| recursive cpu usage | 2,641 | 0.56 | 0.47 |
| redo blocks written | 8,766,094 | 1,854.32 | 1,565.93 |
| redo buffer allocation retries | 24 | 0.01 | 0.00 |
| redo entries | 4,707,068 | 995.70 | 840.85 |
| redo log space requests | 34 | 0.01 | 0.01 |
| redo log space wait time | 50 | 0.01 | 0.01 |
| redo ordering marks | 277,042 | 58.60 | 49.49 |
| redo size | 4,343,559,400 | 918,805.72 | 775,912.72 |
| redo subscn max counts | 2,693 | 0.57 | 0.48 |
| redo synch time | 408 | 0.09 | 0.07 |
| redo synch writes | 6,984 | 1.48 | 1.25 |
| redo wastage | 1,969,620 | 416.64 | 351.84 |
| redo write time | 5,090 | 1.08 | 0.91 |
| redo writer latching time | 1 | 0.00 | 0.00 |
| redo writes | 5,494 | 1.16 | 0.98 |
| rollback changes - undo records applied | 166,609 | 35.24 | 29.76 |
| rollbacks only - consistent read gets | 1,463 | 0.31 | 0.26 |
| rows fetched via callback | 342,159 | 72.38 | 61.12 |
| session connect time | 1,461 | 0.31 | 0.26 |
| session cursor cache hits | 180,472 | 38.18 | 32.24 |
| session logical reads | 16,648,792 | 3,521.77 | 2,974.06 |
| session pga memory | 37,393,448 | 7,909.94 | 6,679.79 |
| session pga memory max | 45,192,232 | 9,559.64 | 8,072.92 |
| session uga memory | 30,067,312,240 | 6,360,225.77 | 5,371,081.14 |
| session uga memory max | 61,930,448 | 13,100.33 | 11,062.96 |
| shared hash latch upgrades - no wait | 6,364 | 1.35 | 1.14 |
| shared hash latch upgrades - wait | 0 | 0.00 | 0.00 |
| sorts (disk) | 4 | 0.00 | 0.00 |
| 磁盘排序一般不能超过5％。如果超过5％，需要设置参数PGA\_AGGREGATE\_TARGET或者SORT\_AREA\_SIZE，注意，这里SORT\_AREA\_SIZE是分配给每个用户的，PGA\_AGGREGATE\_TARGET则是针对所有的session的一个总数设置。 | | | |
| sorts (memory) | 2,857 | 0.60 | 0.51 |
| 内存中的排序数量 | | | |
| sorts (rows) | 42,379,505 | 8,964.66 | 7,570.47 |
| space was found by tune down | 0 | 0.00 | 0.00 |
| space was not found by tune down | 0 | 0.00 | 0.00 |
| sql area evicted | 7 | 0.00 | 0.00 |
| sql area purged | 44 | 0.01 | 0.01 |
| steps of tune down ret. in space pressure | 0 | 0.00 | 0.00 |
| summed dirty queue length | 35,067 | 7.42 | 6.26 |
| switch current to new buffer | 17 | 0.00 | 0.00 |
| table fetch by rowid | 680,469 | 143.94 | 121.56 |
| 这是通过索引或者where rowid=语句来取得的行数，当然这个值越大越好。 | | | |
| table fetch continued row | 0 | 0.00 | 0.00 |
| 这是发生行迁移的行。当行迁移的情况比较严重时，需要对这部分进行优化。  检查行迁移的方法：  1）  运行$ORACLE\_HOME/rdbms/admin/utlchain.sql  2）  analyze table table\_name list chained rows into CHAINED\_ROWS  3）  select \* from CHAINED\_ROWS where table\_name='table\_name';  清除的方法：  方法1：create table table\_name\_tmp as select \* from table\_name where rowed in (select head\_rowid from chained\_rows);         Delete from table\_name where rowed in (select head\_rowid from chained\_rows);         Insert into table\_name select \* from table\_name\_tmp;  方法2：create table table\_name\_tmp select \* from table\_name ;  truncate table table\_name  insert into table\_name select \* from table\_name\_tmp  方法3：用exp工具导出表，然后删除这个表，最后用imp工具导入这表  方法4：alter table table\_name move tablespace tablespace\_name，然后再重新表的索引  上面的4种方法可以用以消除已经存在的行迁移现象，但是行迁移的产生很多情况下时由于PCT\_FREE参数设置的太小所导致，所以需要调整PCT\_FREE参数的值。 | | | |
| table scan blocks gotten | 790,986 | 167.32 | 141.30 |
| table scan rows gotten | 52,989,363 | 11,208.99 | 9,465.77 |
| table scans (long tables) | 4 | 0.00 | 0.00 |
| longtables就是表的大小超过buffer buffer\* \_SMALL\_TABLE\_THRESHOLD的表。如果一个数据库的大表扫描过多，那么db file scattered read等待事件可能同样非常显著。如果table scans (long tables)的per Trans值大于0，你可能需要增加适当的索引来优化你的SQL语句 | | | |
| table scans (short tables) | 169,201 | 35.79 | 30.23 |
| short tables是指表的长度低于buffer chache 2％（2％是有隐含参数\_SMALL\_TABLE\_THRESHOLD定义的，这个参数在oracle不同的版本中，有不同的含义。在9i和10g中，该参数值定义为2%，在8i中，该参数值为20个blocks，在v7中，该参数为5个blocks）的表。这些表将优先使用全表扫描。一般不使用索引。\_SMALL\_TABLE\_THRESHOLD值的计算方法如下（9i,8K）： (db\_cache\_size/8192)\*2％。  注意：\_SMALL\_TABLE\_THRESHOLD参数修改是相当危险的操作 | | | |
| total number of times SMON posted | 259 | 0.05 | 0.05 |
| transaction lock background get time | 0 | 0.00 | 0.00 |
| transaction lock background gets | 0 | 0.00 | 0.00 |
| transaction lock foreground requests | 0 | 0.00 | 0.00 |
| transaction lock foreground wait time | 0 | 0.00 | 0.00 |
| transaction rollbacks | 294 | 0.06 | 0.05 |
| tune down retentions in space pressure | 0 | 0.00 | 0.00 |
| undo change vector size | 1,451,085,596 | 306,952.35 | 259,215.00 |
| user I/O wait time | 11,992 | 2.54 | 2.14 |
| user calls | 1,544,383 | 326.69 | 275.88 |
| user commits | 812 | 0.17 | 0.15 |
| user rollbacks | 4,786 | 1.01 | 0.85 |
| workarea executions - onepass | 1 | 0.00 | 0.00 |
| workarea executions - optimal | 1,616 | 0.34 | 0.29 |
| write clones created in background | 0 | 0.00 | 0.00 |
| write clones created in foreground | 11 | 0.00 | 0.00 |

[**Back to Instance Activity Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#23)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Instance Activity Stats - Absolute Values

* **Statistics with absolute values (should not be diffed)**

|  |  |  |
| --- | --- | --- |
| **Statistic** | **Begin Value** | **End Value** |
| session cursor cache count | 3,024 | 3,592 |
| opened cursors current | 37 | 39 |
| logons current | 24 | 26 |

[**Back to Instance Activity Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#23)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Instance Activity Stats - Thread Activity

* **Statistics identified by '(derived)' come from sources other than SYSSTAT**

|  |  |  |
| --- | --- | --- |
| **Statistic** | **Total** | **per Hour** |
| log switches (derived) | 9 | 6.85 |

[**Back to Instance Activity Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#23)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

## IO Stats

* [**Tablespace IO Stats**](http://blog.itpub.net/26736162/viewspace-1869086/#1700)
* [**File IO Stats**](http://blog.itpub.net/26736162/viewspace-1869086/#1800)

[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

通常，在这里期望在各设备上的读取和写入操作是均匀分布的。要找出什么文件可能非常“热”。一旦DBA了解了如何读取和写入这些数据，他们也许能够通过磁盘间更均匀的分配I/O而得到某些性能提升。

在这里主要关注Av Rd(ms)列 (reads per millisecond)的值，一般来说，大部分的磁盘系统的这个值都能调整到14ms以下，oracle认为该值超过20ms都是不必要的。如果该值超过1000ms，基本可以肯定存在I/O的性能瓶颈。如果在这一列上出现######，可能是你的系统存在严重的I/O问题，也可能是格式的显示问题。

       当出现上面的问题，我们可以考虑以下的方法：

       1）优化操作该表空间或者文件的相关的语句。

       2）如果该表空间包含了索引，可以考虑压缩索引，是索引的分布空间减小，从而减小I/O。

       3）将该表空间分散在多个逻辑卷中，平衡I/O的负载。

       4）我们可以通过设置参数DB\_FILE\_MULTIBLOCK\_READ\_COUNT来调整读取的并行度，这将提高全表扫描的效率。但是也会带来一个问题，就是oracle会因此更多的使用全表扫描而放弃某些索引的使用。为解决这个问题，我们需要设置另外一个参数OPTIMIZER\_INDEX\_COST\_ADJ=30（一般建议设置10－50）。

       关于OPTIMIZER\_INDEX\_COST\_ADJ＝n：该参数是一个百分比值，缺省值为100，可以理解为FULL SCAN COST/INDEX SCAN COST。当n%\* INDEX SCAN COST<FULL SCAN COST时，oracle会选择使用索引。在具体设置的时候，我们可以根据具体的语句来调整该值。如果我们希望某个statement使用索引，而实际它确走全表扫描，可以对比这两种情况的执行计划不同的COST，从而设置一个更合适的值。

       5）检查并调整I/O设备的性能。

### Tablespace IO Stats

* **ordered by IOs (Reads + Writes) desc**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Tablespace** | **Reads** | **Av Reads/s** | **Av Rd(ms)** | **Av Blks/Rd** | **Writes** | **Av Writes/s** | **Buffer Waits** | **Av Buf Wt(ms)** |
| ICCIDAT01 | 67,408 | 14 | 3.76 | 3.17 | 160,261 | 34 | 6 | 0.00 |
| UNDOTBS1 | 10 | 0 | 12.00 | 1.00 | 57,771 | 12 | 625 | 0.02 |
| TEMP | 15,022 | 3 | 8.74 | 7.24 | 3,831 | 1 | 0 | 0.00 |
| USERS | 68 | 0 | 5.44 | 1.00 | 971 | 0 | 0 | 0.00 |
| SYSAUX | 263 | 0 | 5.48 | 1.00 | 458 | 0 | 0 | 0.00 |
| SYSTEM | 32 | 0 | 5.94 | 1.00 | 158 | 0 | 3 | 23.33 |
| UNDOTBS2 | 6 | 0 | 16.67 | 1.00 | 6 | 0 | 0 | 0.00 |

显示每个表空间的I/O统计。根据Oracle经验，Av Rd(ms) [Average Reads in milliseconds]不应该超过30，否则认为有I/O争用。

[**Back to IO Stats**](http://blog.itpub.net/26736162/viewspace-1869086/#24)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### File IO Stats

* **ordered by Tablespace, File**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Tablespace** | **Filename** | **Reads** | **Av Reads/s** | **Av Rd(ms)** | **Av Blks/Rd** | **Writes** | **Av Writes/s** | **Buffer Waits** | **Av Buf Wt(ms)** |
| ICCIDAT01 | /dev/rora\_icci01 | 5,919 | 1 | 4.30 | 3.73 | 15,161 | 3 | 1 | 0.00 |
| ICCIDAT01 | /dev/rora\_icci02 | 7,692 | 2 | 4.12 | 3.18 | 16,555 | 4 | 0 | 0.00 |
| ICCIDAT01 | /dev/rora\_icci03 | 6,563 | 1 | 2.59 | 3.80 | 15,746 | 3 | 0 | 0.00 |
| ICCIDAT01 | /dev/rora\_icci04 | 8,076 | 2 | 2.93 | 3.11 | 16,164 | 3 | 0 | 0.00 |
| ICCIDAT01 | /dev/rora\_icci05 | 6,555 | 1 | 2.61 | 3.31 | 21,958 | 5 | 0 | 0.00 |
| ICCIDAT01 | /dev/rora\_icci06 | 6,943 | 1 | 4.03 | 3.41 | 20,574 | 4 | 0 | 0.00 |
| ICCIDAT01 | /dev/rora\_icci07 | 7,929 | 2 | 4.12 | 2.87 | 18,263 | 4 | 0 | 0.00 |
| ICCIDAT01 | /dev/rora\_icci08 | 7,719 | 2 | 3.83 | 2.99 | 17,361 | 4 | 0 | 0.00 |
| ICCIDAT01 | /dev/rora\_icci09 | 6,794 | 1 | 4.79 | 3.29 | 18,425 | 4 | 0 | 0.00 |
| ICCIDAT01 | /dev/rora\_icci10 | 211 | 0 | 5.31 | 1.00 | 6 | 0 | 0 | 0.00 |
| ICCIDAT01 | /dev/rora\_icci11 | 1,168 | 0 | 4.45 | 1.00 | 6 | 0 | 0 | 0.00 |
| ICCIDAT01 | /dev/rora\_icci12 | 478 | 0 | 4.23 | 1.00 | 6 | 0 | 0 | 0.00 |
| ICCIDAT01 | /dev/rora\_icci13 | 355 | 0 | 5.13 | 1.00 | 6 | 0 | 0 | 0.00 |
| ICCIDAT01 | /dev/rora\_icci14 | 411 | 0 | 4.91 | 1.00 | 6 | 0 | 1 | 0.00 |
| ICCIDAT01 | /dev/rora\_icci15 | 172 | 0 | 5.29 | 1.00 | 6 | 0 | 1 | 0.00 |
| ICCIDAT01 | /dev/rora\_icci16 | 119 | 0 | 7.23 | 1.00 | 6 | 0 | 1 | 0.00 |
| ICCIDAT01 | /dev/rora\_icci17 | 227 | 0 | 6.26 | 1.00 | 6 | 0 | 1 | 0.00 |
| ICCIDAT01 | /dev/rora\_icci18 | 77 | 0 | 8.44 | 1.00 | 6 | 0 | 1 | 0.00 |
| SYSAUX | /dev/rora\_SYSAUX | 263 | 0 | 5.48 | 1.00 | 458 | 0 | 0 | 0.00 |
| SYSTEM | /dev/rora\_SYSTEM | 32 | 0 | 5.94 | 1.00 | 158 | 0 | 3 | 23.33 |
| TEMP | /dev/rora\_TEMP | 3,653 | 1 | 5.67 | 6.61 | 827 | 0 | 0 |  |
| TEMP | /dev/rora\_TEMP2 | 2,569 | 1 | 4.42 | 6.70 | 556 | 0 | 0 |  |
| TEMP | /dev/rora\_TEMP3 | 1,022 | 0 | 2.50 | 16.86 | 557 | 0 | 0 |  |
| TEMP | /dev/rora\_TEMP5 | 7,778 | 2 | 12.43 | 6.46 | 1,891 | 0 | 0 |  |
| UNDOTBS1 | /dev/rora\_UNDO0101 | 10 | 0 | 12.00 | 1.00 | 57,771 | 12 | 625 | 0.02 |
| UNDOTBS2 | /dev/rora\_UNDO0201 | 6 | 0 | 16.67 | 1.00 | 6 | 0 | 0 | 0.00 |
| USERS | /dev/rora\_USERS | 68 | 0 | 5.44 | 1.00 | 971 | 0 | 0 | 0.00 |

[**Back to IO Stats**](http://blog.itpub.net/26736162/viewspace-1869086/#24)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

## Buffer Pool Statistics

* **Standard block size Pools D: default, K: keep, R: recycle**
* **Default Pools for other block sizes: 2k, 4k, 8k, 16k, 32k**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **P** | **Number of Buffers** | **Pool Hit%** | **Buffer Gets** | **Physical Reads** | **Physical Writes** | **Free Buff Wait** | **Writ Comp Wait** | **Buffer Busy Waits** |
| D | 401,071 | 99 | 15,480,754 | 213,729 | 437,340 | 0 | 0 | 634 |

这里将buffer poll细分，列举default、keep、recycle三种类型的buffer的详细情况。在这份报告中，我们的系统中只使用Default size的buffer pool。这里的3个waits统计，其实在前面的等待时间中已经包含，所以可以参考前面的描述。关于命中率也已经在前面讨论。所以，其实这段信息不需要怎么关注。 **[Back to Top](http://blog.itpub.net/26736162/viewspace-1869086/" \l "top" \t "_blank)**

## Advisory Statistics

* [**Instance Recovery Stats**](http://blog.itpub.net/26736162/viewspace-1869086/#2000)
* [**Buffer Pool Advisory**](http://blog.itpub.net/26736162/viewspace-1869086/#2100)
* [**PGA Aggr Summary**](http://blog.itpub.net/26736162/viewspace-1869086/#2400)
* [**PGA Aggr Target Stats**](http://blog.itpub.net/26736162/viewspace-1869086/#2600)
* [**PGA Aggr Target Histogram**](http://blog.itpub.net/26736162/viewspace-1869086/#2700)
* [**PGA Memory Advisory**](http://blog.itpub.net/26736162/viewspace-1869086/#2800)
* [**Shared Pool Advisory**](http://blog.itpub.net/26736162/viewspace-1869086/#5500)
* [**SGA Target Advisory**](http://blog.itpub.net/26736162/viewspace-1869086/#6900)
* [**Streams Pool Advisory**](http://blog.itpub.net/26736162/viewspace-1869086/#7100)
* [**Java Pool Advisory**](http://blog.itpub.net/26736162/viewspace-1869086/#5700)

[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Instance Recovery Stats

* **B: Begin snapshot, E: End snapshot**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Targt MTTR (s)** | **Estd MTTR (s)** | **Recovery Estd IOs** | **Actual Redo Blks** | **Target Redo Blks** | **Log File Size Redo Blks** | **Log Ckpt Timeout Redo Blks** | **Log Ckpt Interval Redo Blks** |
| B | 0 | 11 | 369 | 2316 | 5807 | 1883700 | 5807 |  |
| E | 0 | 98 | 116200 | 1828613 | 1883700 | 1883700 | 5033355 |  |

[**Back to Advisory Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#26)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Buffer Pool Advisory

* **Only rows with estimated physical reads >0 are displayed**
* **ordered by Block Size, Buffers For Estimate**

这是oracle的对buffer pool的大小的调整建议。从advisory的数据看，当然buffer是越大，物理读更小，随着buffer的增大，对物理读的性能改进越来越小。当前buffer 设置为5,120M，物理读因子=1。我们可以看到，buffer pool在3G之前的扩大，对物理读的改善非常明显，之后，这种改善的程度越来越低。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **P** | **Size for Est (M)** | **Size Factor** | **Buffers for Estimate** | **Est Phys Read Factor** | **Estimated Physical Reads** |
| D | 320 | 0.10 | 38,380 | 1.34 | 10,351,726 |
| D | 640 | 0.19 | 76,760 | 1.25 | 9,657,000 |
| D | 960 | 0.29 | 115,140 | 1.08 | 8,365,242 |
| D | 1,280 | 0.38 | 153,520 | 1.04 | 8,059,415 |
| D | 1,600 | 0.48 | 191,900 | 1.02 | 7,878,202 |
| D | 1,920 | 0.57 | 230,280 | 1.01 | 7,841,140 |
| D | 2,240 | 0.67 | 268,660 | 1.01 | 7,829,141 |
| D | 2,560 | 0.77 | 307,040 | 1.01 | 7,817,370 |
| D | 2,880 | 0.86 | 345,420 | 1.01 | 7,804,884 |
| D | 3,200 | 0.96 | 383,800 | 1.00 | 7,784,014 |
| D | 3,344 | 1.00 | 401,071 | 1.00 | 7,748,403 |
| D | 3,520 | 1.05 | 422,180 | 0.99 | 7,702,243 |
| D | 3,840 | 1.15 | 460,560 | 0.99 | 7,680,429 |
| D | 4,160 | 1.24 | 498,940 | 0.99 | 7,663,046 |
| D | 4,480 | 1.34 | 537,320 | 0.99 | 7,653,232 |
| D | 4,800 | 1.44 | 575,700 | 0.99 | 7,645,544 |
| D | 5,120 | 1.53 | 614,080 | 0.98 | 7,630,008 |
| D | 5,440 | 1.63 | 652,460 | 0.98 | 7,616,886 |
| D | 5,760 | 1.72 | 690,840 | 0.98 | 7,614,591 |
| D | 6,080 | 1.82 | 729,220 | 0.98 | 7,613,191 |
| D | 6,400 | 1.91 | 767,600 | 0.98 | 7,599,930 |

[**Back to Advisory Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#26)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### PGA Aggr Summary

* **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** |
| 87.91 | 1,100 | 151 |

[**Back to Advisory Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#26)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### PGA Aggr Target Stats

* **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 Aggr Target(M)** | **Auto PGA Target(M)** | **PGA Mem Alloc(M)** | **W/A PGA Used(M)** | **%PGA W/A Mem** | **%Auto W/A Mem** | **%Man W/A Mem** | **Global Mem Bound(K)** |
| B | 1,024 | 862 | 150.36 | 0.00 | 0.00 | 0.00 | 0.00 | 104,850 |
| E | 1,024 | 860 | 154.14 | 0.00 | 0.00 | 0.00 | 0.00 | 104,850 |

[**Back to Advisory Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#26)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### PGA Aggr Target Histogram

* **Optimal Executions are purely in-memory operations**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Low Optimal** | **High Optimal** | **Total Execs** | **Optimal Execs** | **1-Pass Execs** | **M-Pass Execs** |
| 2K | 4K | 1,385 | 1,385 | 0 | 0 |
| 64K | 128K | 28 | 28 | 0 | 0 |
| 128K | 256K | 5 | 5 | 0 | 0 |
| 256K | 512K | 79 | 79 | 0 | 0 |
| 512K | 1024K | 108 | 108 | 0 | 0 |
| 1M | 2M | 7 | 7 | 0 | 0 |
| 8M | 16M | 1 | 1 | 0 | 0 |
| 128M | 256M | 3 | 2 | 1 | 0 |
| 256M | 512M | 1 | 1 | 0 | 0 |

[**Back to Advisory Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#26)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### PGA Memory Advisory

* **When using Auto Memory Mgmt, minimally choose a pga\_aggregate\_target value where Estd PGA Overalloc Count is 0**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **PGA Target Est (MB)** | **Size Factr** | **W/A MB Processed** | **Estd Extra W/A MB Read/ Written to Disk** | **Estd PGA Cache Hit %** | **Estd PGA Overalloc Count** |
| 128 | 0.13 | 4,652.12 | 2,895.99 | 62.00 | 0 |
| 256 | 0.25 | 4,652.12 | 2,857.13 | 62.00 | 0 |
| 512 | 0.50 | 4,652.12 | 2,857.13 | 62.00 | 0 |
| 768 | 0.75 | 4,652.12 | 2,857.13 | 62.00 | 0 |
| 1,024 | 1.00 | 4,652.12 | 717.82 | 87.00 | 0 |
| 1,229 | 1.20 | 4,652.12 | 717.82 | 87.00 | 0 |
| 1,434 | 1.40 | 4,652.12 | 717.82 | 87.00 | 0 |
| 1,638 | 1.60 | 4,652.12 | 717.82 | 87.00 | 0 |
| 1,843 | 1.80 | 4,652.12 | 717.82 | 87.00 | 0 |
| 2,048 | 2.00 | 4,652.12 | 717.82 | 87.00 | 0 |
| 3,072 | 3.00 | 4,652.12 | 717.82 | 87.00 | 0 |
| 4,096 | 4.00 | 4,652.12 | 717.82 | 87.00 | 0 |
| 6,144 | 6.00 | 4,652.12 | 717.82 | 87.00 | 0 |
| 8,192 | 8.00 | 4,652.12 | 717.82 | 87.00 | 0 |

[**Back to Advisory Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#26)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Shared Pool Advisory

* **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.**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Shared Pool Size(M)** | **SP Size Factr** | **Est LC Size (M)** | **Est LC Mem Obj** | **Est LC Time Saved (s)** | **Est LC Time Saved Factr** | **Est LC Load Time (s)** | **Est LC Load Time Factr** | **Est LC Mem Obj Hits** |
| 304 | 0.43 | 78 | 7,626 | 64,842 | 1.00 | 31 | 1.00 | 3,206,955 |
| 384 | 0.55 | 78 | 7,626 | 64,842 | 1.00 | 31 | 1.00 | 3,206,955 |
| 464 | 0.66 | 78 | 7,626 | 64,842 | 1.00 | 31 | 1.00 | 3,206,955 |
| 544 | 0.77 | 78 | 7,626 | 64,842 | 1.00 | 31 | 1.00 | 3,206,955 |
| 624 | 0.89 | 78 | 7,626 | 64,842 | 1.00 | 31 | 1.00 | 3,206,955 |
| 704 | 1.00 | 78 | 7,626 | 64,842 | 1.00 | 31 | 1.00 | 3,206,955 |
| 784 | 1.11 | 78 | 7,626 | 64,842 | 1.00 | 31 | 1.00 | 3,206,955 |
| 864 | 1.23 | 78 | 7,626 | 64,842 | 1.00 | 31 | 1.00 | 3,206,955 |
| 944 | 1.34 | 78 | 7,626 | 64,842 | 1.00 | 31 | 1.00 | 3,206,955 |
| 1,024 | 1.45 | 78 | 7,626 | 64,842 | 1.00 | 31 | 1.00 | 3,206,955 |
| 1,104 | 1.57 | 78 | 7,626 | 64,842 | 1.00 | 31 | 1.00 | 3,206,955 |
| 1,184 | 1.68 | 78 | 7,626 | 64,842 | 1.00 | 31 | 1.00 | 3,206,955 |
| 1,264 | 1.80 | 78 | 7,626 | 64,842 | 1.00 | 31 | 1.00 | 3,206,955 |
| 1,344 | 1.91 | 78 | 7,626 | 64,842 | 1.00 | 31 | 1.00 | 3,206,955 |
| 1,424 | 2.02 | 78 | 7,626 | 64,842 | 1.00 | 31 | 1.00 | 3,206,955 |

[**Back to Advisory Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#26)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### SGA Target Advisory

|  |  |  |  |
| --- | --- | --- | --- |
| **SGA Target Size (M)** | **SGA Size Factor** | **Est DB Time (s)** | **Est Physical Reads** |
| 1,024 | 0.25 | 9,060 | 9,742,760 |
| 2,048 | 0.50 | 7,612 | 7,948,245 |
| 3,072 | 0.75 | 7,563 | 7,886,258 |
| 4,096 | 1.00 | 7,451 | 7,748,338 |
| 5,120 | 1.25 | 7,423 | 7,713,470 |
| 6,144 | 1.50 | 7,397 | 7,680,927 |
| 7,168 | 1.75 | 7,385 | 7,666,980 |
| 8,192 | 2.00 | 7,385 | 7,666,980 |

[**Back to Advisory Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#26)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Streams Pool Advisory

**No data exists for this section of the report.**

[**Back to Advisory Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#26)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Java Pool Advisory

**No data exists for this section of the report.**

[**Back to Advisory Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#26)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

## Wait Statistics

* [**Buffer Wait Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#2300)
* [**Enqueue Activity**](http://blog.itpub.net/26736162/viewspace-1869086/#3000)

[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Buffer Wait Statistics

* **ordered by wait time desc, waits desc**

|  |  |  |  |
| --- | --- | --- | --- |
| **Class** | **Waits** | **Total Wait Time (s)** | **Avg Time (ms)** |
| data block | 3 | 0 | 23 |
| undo header | 616 | 0 | 0 |
| file header block | 8 | 0 | 0 |
| undo block | 7 | 0 | 0 |

[**Back to Wait Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#27)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Enqueue Activity

* **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)** |
| FB-Format Block | 14,075 | 14,075 | 0 | 7,033 | 3 | 0.43 |
| US-Undo Segment | 964 | 964 | 0 | 556 | 0 | 0.32 |
| WF-AWR Flush | 24 | 24 | 0 | 14 | 0 | 9.00 |
| HW-Segment High Water Mark | 4,223 | 4,223 | 0 | 37 | 0 | 1.22 |
| CF-Controlfile Transaction | 10,548 | 10,548 | 0 | 58 | 0 | 0.67 |
| TX-Transaction (index contention) | 1 | 1 | 0 | 1 | 0 | 35.00 |
| TM-DML | 121,768 | 121,761 | 6 | 70 | 0 | 0.43 |
| PS-PX Process Reservation | 103 | 103 | 0 | 46 | 0 | 0.65 |
| TT-Tablespace | 9,933 | 9,933 | 0 | 39 | 0 | 0.54 |
| TD-KTF map table enqueue (KTF dump entries) | 12 | 12 | 0 | 12 | 0 | 1.42 |
| TA-Instance Undo | 18 | 18 | 0 | 13 | 0 | 0.38 |
| PI-Remote PX Process Spawn Status | 16 | 16 | 0 | 8 | 0 | 0.50 |
| MW-MWIN Schedule | 3 | 3 | 0 | 3 | 0 | 0.67 |
| DR-Distributed Recovery | 3 | 3 | 0 | 3 | 0 | 0.33 |
| TS-Temporary Segment | 14 | 11 | 3 | 3 | 0 | 0.33 |
| AF-Advisor Framework (task serialization) | 14 | 14 | 0 | 1 | 0 | 1.00 |
| JS-Job Scheduler (job run lock - synchronize) | 2 | 2 | 0 | 1 | 0 | 1.00 |
| UL-User-defined | 2 | 2 | 0 | 1 | 0 | 1.00 |
| MD-Materialized View Log DDL | 6 | 6 | 0 | 2 | 0 | 0.00 |

[**Back to Wait Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#27)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

## Undo Statistics

* [**Undo Segment Summary**](http://blog.itpub.net/26736162/viewspace-1869086/#3300)
* [**Undo Segment Stats**](http://blog.itpub.net/26736162/viewspace-1869086/#3400)

[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

Undo从9i开始，回滚段一般都是自动管理的，一般情况下，这里我们不需要太重点关注。

在这里，主要关注pct waits，如果出现比较多的pct waits，那就需要增加回滚段的数量或者增大回滚段的空间。另外，观察一下各个回滚段使用的情况，比较理想的是各个回滚段上Avg Active比较均衡。

在oracle 9i之前，回滚段时手工管理的，可以通过指定optimal值来设定一个回滚段收缩的值，如果不设定，默认也应当为initial+(minextents-1)\*next extents ，这个指定的结果，就是限制了回滚段不能无限制的增长，当超过optimal的设定值后，在适当的时候，oracle会shrinks到optimal大小。但是9i之后，undo一般都设置为auto模式，在这种模式下，我们无法指定optimal值，好像也没有默认值，所以无法shrinks，回滚段就会无限制的增长，一直到表空间利用率达到为100％，如果表空间设置为自动扩展的方式，这种情况下，就更糟糕，undo将无限制的增长。在这里，我们也可以看到，shrinks的值为0，也就是说，从来就没收缩过。

### Segment Summary

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

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Undo TS#** | **Num Undo Blocks (K)** | **Number of Transactions** | **Max Qry Len (s)** | **Max Tx Concurcy** | **Min/Max TR (mins)** | **STO/ OOS** | **uS/uR/uU/ eS/eR/eU** |
| 1 | 219.12 | 113,405 | 0 | 6 | 130.95/239.25 | 0/0 | 0/0/0/13/24256/0 |

[**Back to Undo Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#28)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Undo Segment Stats

* **Most recent 35 Undostat rows, ordered by Time desc**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **End Time** | **Num Undo Blocks** | **Number of Transactions** | **Max Qry Len (s)** | **Max Tx Concy** | **Tun Ret (mins)** | **STO/ OOS** | **uS/uR/uU/ eS/eR/eU** |
| 25-Dec 15:18 | 182,021 | 74,309 | 0 | 5 | 131 | 0/0 | 0/0/0/13/24256/0 |
| 25-Dec 15:08 | 57 | 170 | 0 | 3 | 239 | 0/0 | 0/0/0/0/0/0 |
| 25-Dec 14:58 | 68 | 31 | 0 | 2 | 229 | 0/0 | 0/0/0/0/0/0 |
| 25-Dec 14:48 | 194 | 4,256 | 0 | 4 | 219 | 0/0 | 0/0/0/0/0/0 |
| 25-Dec 14:38 | 570 | 12,299 | 0 | 5 | 209 | 0/0 | 0/0/0/0/0/0 |
| 25-Dec 14:28 | 36,047 | 21,328 | 0 | 6 | 200 | 0/0 | 0/0/0/0/0/0 |
| 25-Dec 14:18 | 70 | 907 | 0 | 3 | 162 | 0/0 | 0/0/0/0/0/0 |
| 25-Dec 14:08 | 91 | 105 | 0 | 3 | 154 | 0/0 | 0/0/0/0/0/0 |

[**Back to Undo Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#28)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

## Latch Statistics

* [**Latch Activity**](http://blog.itpub.net/26736162/viewspace-1869086/#3500)
* [**Latch Sleep Breakdown**](http://blog.itpub.net/26736162/viewspace-1869086/#3600)
* [**Latch Miss Sources**](http://blog.itpub.net/26736162/viewspace-1869086/#3700)
* [**Parent Latch Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#3800)
* [**Child Latch Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#3900)

[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

Latch是一种低级排队机制，用于防止对内存结构的并行访问，保护系统全局区(SGA)共享内存结构。Latch是一种快速地被获取和释放的内存锁。如果latch不可用，就会记录latch free miss 。

有两种类型的Latch:willing to wait和（immediate）not willing to wait。

对于愿意等待类型(willing-to-wait)的latch,如果一个进程在第一次尝试中没有获得latch,那么它会等待并且再尝试一次,如果经过\_spin\_count次争夺不能获得latch, 然后该进程转入睡眠状态，百分之一秒之后醒来，按顺序重复以前的步骤。在8i/9i中默认值是\_spin\_count=2000。睡眠的时间会越来越长。

　　对于不愿意等待类型(not-willing-to-wait)的latch，如果该闩不能立即得到的话，那么该进程就不会为获得该闩而等待。它将继续执行另一个操作。

　　大多数Latch问题都可以归结为以下几种:

　　没有很好的是用绑定变量(library cache latch和shared pool cache)、重作生成问题(redo allocation latch)、缓冲存储竞争问题(cache buffers LRU chain)，以及buffer cache中的存在"热点"块(cache buffers chain)。

另外也有一些latch等待与bug有关，应当关注Metalink相关bug的公布及补丁的发布。

当latch miss ratios大于0.5%时，就需要检查latch的等待问题。

如果SQL语句不能调整，在8.1.6版本以上，可以通过设置CURSOR\_SHARING = force 在服务器端强制绑定变量。设置该参数可能会带来一定的副作用，可能会导致执行计划不优，另外对于Java的程序，有相关的bug，具体应用应该关注Metalink的bug公告。

下面对几个重要类型的latch等待加以说明：

1)    latch free：当‘latch free’在报告的高等待事件中出现时，就表示可能出现了性能问题，就需要在这一部分详细分析出现等待的具体的latch的类型，然后再调整。

2)    cache buffers chain：cbc latch表明热块。为什么这会表示存在热块？为了理解这个问题，先要理解cbc的作用。ORACLE对buffer cache管理是以hash链表的方式来实现的（oracle称为buckets，buckets的数量由\_db\_block\_hash\_buckets定义）。cbc latch就是为了保护buffer cache而设置的。当有并发的访问需求时，cbc会将这些访问串行化，当我们获得cbc latch的控制权时，就可以开始访问数据，如果我们所请求的数据正好的某个buckets中，那就直接从内存中读取数据，完成之后释放cbc latch，cbc latch就可以被其他的用户获取了。cbc latch获取和释放是非常快速的，所以这种情况下就一般不会存在等待。但是如果我们请求的数据在内存中不存在，就需要到物理磁盘上读取数据，这相对于latch来说就是一个相当长的时间了，当找到对应的数据块时，如果有其他用户正在访问这个数据块，并且数据块上也没有空闲的ITL槽来接收本次请求，就必须等待。在这过程中，我们因为没有得到请求的数据，就一直占有cbc latch，其他的用户也就无法获取cbc latch，所以就出现了cbc latch等待的情况。所以这种等待归根结底就是由于数据块比较hot的造成的。  
解决方法可以参考前面在等待事件中的3) buffer busy wait中关于热块的解决方法。

3)    cache buffers lru chain：该latch用于扫描buffer的LRU链表。三种情况可导致争用：1）buffer cache太小 ；2）buffer cache的过度使用，或者太多的基于cache的排序操作；3）DBWR不及时。解决方法：查找逻辑读过高的statement，增大buffer cache。

4)    Library cache and shared pool 争用：  
library cache是一个hash table，我们需要通过一个hash buckets数组来访问（类似buffer cache）。library cache latch就是将对library cache的访问串行化。当有一个sql（或者PL/SQL procedure，package，function，trigger）需要执行的时候，首先需要获取一个latch，然后library cache latch就会去查询library cache以重用这些语句。在8i中，library cache latch只有一个。在9i中，有7个child latch，这个数量可以通过参数\_KGL\_LATCH\_ COUNT修改（最大可以达到66个）。当共享池太小或者语句的reuse低的时候，会出现‘shared pool’、‘library cache pin’或者 ‘library cache’ latch的争用。解决的方法是：增大共享池或者设置CURSOR\_SHARING=FORCE|SIMILAR ，当然我们也需要tuning SQL statement。为减少争用，我们也可以把一些比较大的SQL或者过程利用DBMS\_SHARED\_POOL.KEEP包来pinning在shared pool中。  
shared pool内存结构与buffer cache类似，也采用的是hash方式来管理的。共享池有一个固定数量的hash buckets，通过固定数量的library cache latch来串行化保护这段内存的使用。在数据启动的时候，会分配509个hash buctets，2\*CPU\_COUNT个library cache latch。当在数据库的使用中，共享池中的对象越来越多，oracle就会以以下的递增方式增加hash buckets的数量：509,1021,4093,8191,32749,65521,131071,4292967293。我们可以通过设置下面的参数来实现\_KGL\_BUCKET\_COUNT，参数的默认值是0，代表数量509，最大我们可以设置为8，代表数量131071。  
我们可以通过x$ksmsp来查看具体的共享池内存段情况，主要关注下面几个字段：  
KSMCHCOM—表示内存段的类型  
ksmchptr—表示内存段的物理地址  
ksmchsiz—表示内存段的大小  
ksmchcls—表示内存段的分类。recr表示a recreatable piece currently in use that can be a candidate for flushing when the shared pool is low in available memory; freeabl表示当前正在使用的，能够被释放的段； free表示空闲的未分配的段； perm表示不能被释放永久分配段。  
降低共享池的latch 争用，我们主要可以考虑如下的几个事件：  
1、使用绑定变量  
2、使用cursor sharing  
3、设置session\_cached\_cursors参数。该参数的作用是将cursor从shared pool转移到pga中。减小对共享池的争用。一般初始的值可以设置为100，然后视情况再作调整。  
4、设置合适大小的共享池

5)    Redo Copy：这个latch用来从PGA中copy redo records到redo log buffer。latch的初始数量是2\*COU\_OUNT，可以通过设置参数\_LOG\_SIMULTANEOUS\_COPIES在增加latch的数量，减小争用。

6)    Redo allocation：该latch用于redo log buffer的分配。减小这种类型的争用的方法有3个：  
增大redo log buffer  
适当使用nologging选项  
避免不必要的commit操作

7)    Row cache objects：该latch出现争用，通常表明数据字典存在争用的情况，这往往也预示着过多的依赖于公共同义词的parse。解决方法：1）增大shared pool  2）使用本地管理的表空间，尤其对于索引表空间

|  |  |
| --- | --- |
| Latch事件 | 建议解决方法 |
| Library cache | 使用绑定变量; 调整shared\_pool\_size. |
| Shared pool | 使用绑定变量; 调整shared\_pool\_size. |
| Redo allocation | 减小 redo 的产生； 避免不必要的commits. |
| Redo copy | 增加 \_log\_simultaneous\_copies. |
| Row cache objects | 增加shared\_pool\_size |
| Cache buffers chain | 增大 \_DB\_BLOCK\_HASH\_BUCKETS ； make it prime. |
| Cache buffers LRU chain | 使用多个缓冲池；调整引起大量逻辑读的查询 |

注：在这里，提到了不少隐藏参数，也有利用隐藏参数来解决latch的方法描述，但是在实际的操作中，强烈建议尽量不要去更改隐藏参数的默认值。

### Latch Activity

* **"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**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Latch Name** | **Get Requests** | **Pct Get Miss** | **Avg Slps /Miss** | **Wait Time (s)** | **NoWait Requests** | **Pct NoWait Miss** |
| ASM db client latch | 11,883 | 0.00 |  | 0 | 0 |  |
| AWR Alerted Metric Element list | 18,252 | 0.00 |  | 0 | 0 |  |
| Consistent RBA | 5,508 | 0.02 | 0.00 | 0 | 0 |  |
| FOB s.o list latch | 731 | 0.00 |  | 0 | 0 |  |
| JS broadcast add buf latch | 6,193 | 0.00 |  | 0 | 0 |  |
| JS broadcast drop buf latch | 6,194 | 0.00 |  | 0 | 0 |  |
| JS broadcast load blnc latch | 6,057 | 0.00 |  | 0 | 0 |  |
| JS mem alloc latch | 8 | 0.00 |  | 0 | 0 |  |
| JS queue access latch | 8 | 0.00 |  | 0 | 0 |  |
| JS queue state obj latch | 218,086 | 0.00 |  | 0 | 0 |  |
| JS slv state obj latch | 31 | 0.00 |  | 0 | 0 |  |
| KCL gc element parent latch | 2,803,392 | 0.04 | 0.01 | 0 | 108 | 0.00 |
| KJC message pool free list | 43,168 | 0.06 | 0.00 | 0 | 14,532 | 0.01 |
| KJCT flow control latch | 563,875 | 0.00 | 0.00 | 0 | 0 |  |
| KMG MMAN ready and startup request latch | 1,576 | 0.00 |  | 0 | 0 |  |
| KSXR large replies | 320 | 0.00 |  | 0 | 0 |  |
| KTF sga latch | 23 | 0.00 |  | 0 | 1,534 | 0.00 |
| KWQMN job cache list latch | 352 | 0.00 |  | 0 | 0 |  |
| KWQP Prop Status | 5 | 0.00 |  | 0 | 0 |  |
| MQL Tracking Latch | 0 |  |  | 0 | 94 | 0.00 |
| Memory Management Latch | 0 |  |  | 0 | 1,576 | 0.00 |
| OS process | 207 | 0.00 |  | 0 | 0 |  |
| OS process allocation | 1,717 | 0.00 |  | 0 | 0 |  |
| OS process: request allocation | 73 | 0.00 |  | 0 | 0 |  |
| PL/SQL warning settings | 226 | 0.00 |  | 0 | 0 |  |
| SGA IO buffer pool latch | 20,679 | 0.06 | 0.00 | 0 | 20,869 | 0.00 |
| SQL memory manager latch | 7 | 0.00 |  | 0 | 1,575 | 0.00 |
| SQL memory manager workarea list latch | 439,442 | 0.00 |  | 0 | 0 |  |
| Shared B-Tree | 182 | 0.00 |  | 0 | 0 |  |
| Undo Hint Latch | 0 |  |  | 0 | 12 | 0.00 |
| active checkpoint queue latch | 7,835 | 0.00 |  | 0 | 0 |  |
| active service list | 50,936 | 0.00 |  | 0 | 1,621 | 0.00 |
| archive control | 5 | 0.00 |  | 0 | 0 |  |
| begin backup scn array | 72,901 | 0.00 | 0.00 | 0 | 0 |  |
| business card | 32 | 0.00 |  | 0 | 0 |  |
| cache buffer handles | 331,153 | 0.02 | 0.00 | 0 | 0 |  |
| cache buffers chains | 48,189,073 | 0.00 | 0.00 | 0 | 1,201,379 | 0.00 |
| cache buffers lru chain | 891,796 | 0.34 | 0.00 | 0 | 991,605 | 0.23 |
| cache table scan latch | 0 |  |  | 0 | 10,309 | 0.01 |
| channel handle pool latch | 99 | 0.00 |  | 0 | 0 |  |
| channel operations parent latch | 490,324 | 0.01 | 0.00 | 0 | 0 |  |
| checkpoint queue latch | 671,856 | 0.01 | 0.00 | 0 | 555,469 | 0.02 |
| client/application info | 335 | 0.00 |  | 0 | 0 |  |
| commit callback allocation | 12 | 0.00 |  | 0 | 0 |  |
| compile environment latch | 173,428 | 0.00 |  | 0 | 0 |  |
| dml lock allocation | 243,087 | 0.00 | 0.00 | 0 | 0 |  |
| dummy allocation | 134 | 0.00 |  | 0 | 0 |  |
| enqueue hash chains | 1,539,499 | 0.01 | 0.03 | 0 | 263 | 0.00 |
| enqueues | 855,207 | 0.02 | 0.00 | 0 | 0 |  |
| error message lists | 64 | 0.00 |  | 0 | 0 |  |
| event group latch | 38 | 0.00 |  | 0 | 0 |  |
| file cache latch | 4,694 | 0.00 |  | 0 | 0 |  |
| gcs drop object freelist | 8,451 | 0.19 | 0.00 | 0 | 0 |  |
| gcs opaque info freelist | 38,584 | 0.00 | 0.00 | 0 | 0 |  |
| gcs partitioned table hash | 9,801,867 | 0.00 |  | 0 | 0 |  |
| gcs remaster request queue | 31 | 0.00 |  | 0 | 0 |  |
| gcs remastering latch | 1,014,198 | 0.00 | 0.33 | 0 | 0 |  |
| gcs resource freelist | 1,154,551 | 0.03 | 0.00 | 0 | 771,650 | 0.00 |
| gcs resource hash | 3,815,373 | 0.02 | 0.00 | 0 | 2 | 0.00 |
| gcs resource scan list | 4 | 0.00 |  | 0 | 0 |  |
| gcs shadows freelist | 795,482 | 0.00 | 0.00 | 0 | 779,648 | 0.00 |
| ges caches resource lists | 209,655 | 0.02 | 0.00 | 0 | 121,613 | 0.01 |
| ges deadlock list | 840 | 0.00 |  | 0 | 0 |  |
| ges domain table | 366,702 | 0.00 |  | 0 | 0 |  |
| ges enqueue table freelist | 487,875 | 0.00 |  | 0 | 0 |  |
| ges group table | 543,887 | 0.00 |  | 0 | 0 |  |
| ges process hash list | 59,503 | 0.00 |  | 0 | 0 |  |
| ges process parent latch | 908,232 | 0.00 |  | 0 | 1 | 0.00 |
| ges process table freelist | 73 | 0.00 |  | 0 | 0 |  |
| ges resource hash list | 862,590 | 0.02 | 0.28 | 0 | 72,266 | 0.01 |
| ges resource scan list | 534 | 0.00 |  | 0 | 0 |  |
| ges resource table freelist | 135,406 | 0.00 | 0.00 | 0 | 0 |  |
| ges synchronous data | 160 | 0.63 | 0.00 | 0 | 2,954 | 0.07 |
| ges timeout list | 3,256 | 0.00 |  | 0 | 4,478 | 0.00 |
| global KZLD latch for mem in SGA | 21 | 0.00 |  | 0 | 0 |  |
| hash table column usage latch | 59 | 0.00 |  | 0 | 1,279 | 0.00 |
| hash table modification latch | 116 | 0.00 |  | 0 | 0 |  |
| job workq parent latch | 0 |  |  | 0 | 14 | 0.00 |
| job\_queue\_processes parameter latch | 86 | 0.00 |  | 0 | 0 |  |
| kks stats | 384 | 0.00 |  | 0 | 0 |  |
| ksuosstats global area | 329 | 0.00 |  | 0 | 0 |  |
| ktm global data | 296 | 0.00 |  | 0 | 0 |  |
| kwqbsn:qsga | 182 | 0.00 |  | 0 | 0 |  |
| lgwr LWN SCN | 6,547 | 0.18 | 0.00 | 0 | 0 |  |
| library cache | 235,060 | 0.00 | 0.00 | 0 | 22 | 0.00 |
| library cache load lock | 486 | 0.00 |  | 0 | 0 |  |
| library cache lock | 49,284 | 0.00 |  | 0 | 0 |  |
| library cache lock allocation | 566 | 0.00 |  | 0 | 0 |  |
| library cache pin | 27,863 | 0.00 | 0.00 | 0 | 0 |  |
| library cache pin allocation | 204 | 0.00 |  | 0 | 0 |  |
| list of block allocation | 10,101 | 0.00 |  | 0 | 0 |  |
| loader state object freelist | 108 | 0.00 |  | 0 | 0 |  |
| longop free list parent | 6 | 0.00 |  | 0 | 6 | 0.00 |
| message pool operations parent latch | 1,424 | 0.00 |  | 0 | 0 |  |
| messages | 222,581 | 0.00 | 0.00 | 0 | 0 |  |
| mostly latch-free SCN | 6,649 | 1.43 | 0.00 | 0 | 0 |  |
| multiblock read objects | 29,230 | 0.03 | 0.00 | 0 | 0 |  |
| name-service memory objects | 18,842 | 0.00 |  | 0 | 0 |  |
| name-service namespace bucket | 56,712 | 0.00 |  | 0 | 0 |  |
| name-service namespace objects | 15 | 0.00 |  | 0 | 0 |  |
| name-service pending queue | 6,436 | 0.00 |  | 0 | 0 |  |
| name-service request | 44 | 0.00 |  | 0 | 0 |  |
| name-service request queue | 57,312 | 0.00 |  | 0 | 0 |  |
| ncodef allocation latch | 77 | 0.00 |  | 0 | 0 |  |
| object queue header heap | 37,721 | 0.00 |  | 0 | 7,457 | 0.00 |
| object queue header operation | 2,706,992 | 0.06 | 0.00 | 0 | 0 |  |
| object stats modification | 22 | 0.00 |  | 0 | 0 |  |
| parallel query alloc buffer | 939 | 0.00 |  | 0 | 0 |  |
| parallel query stats | 72 | 0.00 |  | 0 | 0 |  |
| parallel txn reco latch | 630 | 0.00 |  | 0 | 0 |  |
| parameter list | 193 | 0.00 |  | 0 | 0 |  |
| parameter table allocation management | 68 | 0.00 |  | 0 | 0 |  |
| post/wait queue | 4,205 | 0.00 |  | 0 | 2,712 | 0.00 |
| process allocation | 46,895 | 0.00 |  | 0 | 38 | 0.00 |
| process group creation | 73 | 0.00 |  | 0 | 0 |  |
| process queue | 175 | 0.00 |  | 0 | 0 |  |
| process queue reference | 2,621 | 0.00 |  | 0 | 240 | 62.50 |
| qmn task queue latch | 668 | 0.15 | 1.00 | 0 | 0 |  |
| query server freelists | 159 | 0.00 |  | 0 | 0 |  |
| query server process | 8 | 0.00 |  | 0 | 7 | 0.00 |
| queued dump request | 23,628 | 0.00 |  | 0 | 0 |  |
| redo allocation | 21,206 | 0.57 | 0.00 | 0 | 4,706,826 | 0.02 |
| redo copy | 0 |  |  | 0 | 4,707,106 | 0.01 |
| redo writing | 29,944 | 0.01 | 0.00 | 0 | 0 |  |
| resmgr group change latch | 69 | 0.00 |  | 0 | 0 |  |
| resmgr:actses active list | 137 | 0.00 |  | 0 | 0 |  |
| resmgr:actses change group | 52 | 0.00 |  | 0 | 0 |  |
| resmgr:free threads list | 130 | 0.00 |  | 0 | 0 |  |
| resmgr:schema config | 7 | 0.00 |  | 0 | 0 |  |
| row cache objects | 1,644,149 | 0.00 | 0.00 | 0 | 321 | 0.00 |
| rules engine rule set statistics | 500 | 0.00 |  | 0 | 0 |  |
| sequence cache | 360 | 0.00 |  | 0 | 0 |  |
| session allocation | 535,514 | 0.00 | 0.00 | 0 | 0 |  |
| session idle bit | 3,262,141 | 0.00 | 0.00 | 0 | 0 |  |
| session state list latch | 166 | 0.00 |  | 0 | 0 |  |
| session switching | 77 | 0.00 |  | 0 | 0 |  |
| session timer | 1,620 | 0.00 |  | 0 | 0 |  |
| shared pool | 60,359 | 0.00 | 0.00 | 0 | 0 |  |
| shared pool sim alloc | 13 | 0.00 |  | 0 | 0 |  |
| shared pool simulator | 4,246 | 0.00 |  | 0 | 0 |  |
| simulator hash latch | 1,862,803 | 0.00 |  | 0 | 0 |  |
| simulator lru latch | 1,719,480 | 0.01 | 0.00 | 0 | 46,053 | 0.00 |
| slave class | 2 | 0.00 |  | 0 | 0 |  |
| slave class create | 8 | 12.50 | 1.00 | 0 | 0 |  |
| sort extent pool | 1,284 | 0.00 |  | 0 | 0 |  |
| state object free list | 4 | 0.00 |  | 0 | 0 |  |
| statistics aggregation | 280 | 0.00 |  | 0 | 0 |  |
| temp lob duration state obj allocation | 2 | 0.00 |  | 0 | 0 |  |
| threshold alerts latch | 202 | 0.00 |  | 0 | 0 |  |
| transaction allocation | 211 | 0.00 |  | 0 | 0 |  |
| transaction branch allocation | 77 | 0.00 |  | 0 | 0 |  |
| undo global data | 779,759 | 0.07 | 0.00 | 0 | 0 |  |
| user lock | 102 | 0.00 |  | 0 | 0 |  |

[**Back to Latch Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#29)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Latch Sleep Breakdown

* **ordered by misses desc**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Latch Name** | **Get Requests** | **Misses** | **Sleeps** | **Spin Gets** | **Sleep1** | **Sleep2** | **Sleep3** |
| cache buffers lru chain | 891,796 | 3,061 | 1 | 3,060 | 0 | 0 | 0 |
| object queue header operation | 2,706,992 | 1,755 | 3 | 1,752 | 0 | 0 | 0 |
| KCL gc element parent latch | 2,803,392 | 1,186 | 11 | 1,176 | 0 | 0 | 0 |
| cache buffers chains | 48,189,073 | 496 | 1 | 495 | 0 | 0 | 0 |
| ges resource hash list | 862,590 | 160 | 44 | 116 | 0 | 0 | 0 |
| enqueue hash chains | 1,539,499 | 79 | 2 | 78 | 0 | 0 | 0 |
| gcs remastering latch | 1,014,198 | 3 | 1 | 2 | 0 | 0 | 0 |
| qmn task queue latch | 668 | 1 | 1 | 0 | 0 | 0 | 0 |
| slave class create | 8 | 1 | 1 | 0 | 0 | 0 | 0 |

[**Back to Latch Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#29)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Latch Miss Sources

* **only latches with sleeps are shown**
* **ordered by name, sleeps desc**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Latch Name** | **Where** | **NoWait Misses** | **Sleeps** | **Waiter Sleeps** |
| KCL gc element parent latch | kclrwrite | 0 | 8 | 0 |
| KCL gc element parent latch | kclnfndnewm | 0 | 4 | 6 |
| KCL gc element parent latch | KCLUNLNK | 0 | 1 | 1 |
| KCL gc element parent latch | kclbla | 0 | 1 | 0 |
| KCL gc element parent latch | kclulb | 0 | 1 | 1 |
| KCL gc element parent latch | kclzcl | 0 | 1 | 0 |
| cache buffers chains | kcbnew: new latch again | 0 | 2 | 0 |
| cache buffers chains | kclwrt | 0 | 1 | 0 |
| cache buffers lru chain | kcbzgws | 0 | 1 | 0 |
| enqueue hash chains | ksqcmi: if lk mode not requested | 0 | 2 | 0 |
| event range base latch | No latch | 0 | 1 | 1 |
| gcs remastering latch | 69 | 0 | 1 | 0 |
| ges resource hash list | kjlmfnd: search for lockp by rename and inst id | 0 | 23 | 0 |
| ges resource hash list | kjakcai: search for resp by resname | 0 | 13 | 0 |
| ges resource hash list | kjrmas1: lookup master node | 0 | 5 | 0 |
| ges resource hash list | kjlrlr: remove lock from resource queue | 0 | 2 | 33 |
| ges resource hash list | kjcvscn: remove from scan queue | 0 | 1 | 0 |
| object queue header operation | kcbo\_switch\_q\_bg | 0 | 3 | 0 |
| object queue header operation | kcbo\_switch\_mq\_bg | 0 | 2 | 4 |
| object queue header operation | kcbw\_unlink\_q | 0 | 2 | 0 |
| object queue header operation | kcbw\_link\_q | 0 | 1 | 0 |
| slave class create | ksvcreate | 0 | 1 | 0 |

[**Back to Latch Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#29)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Parent Latch Statistics

**No data exists for this section of the report.**

[**Back to Latch Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#29)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Child Latch Statistics

**No data exists for this section of the report.**

[**Back to Latch Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#29)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

## Segment Statistics

* [**Segments by Logical Reads**](http://blog.itpub.net/26736162/viewspace-1869086/#4000)
* [**Segments by Physical Reads**](http://blog.itpub.net/26736162/viewspace-1869086/#4100)
* [**Segments by Row Lock Waits**](http://blog.itpub.net/26736162/viewspace-1869086/#4200)
* [**Segments by ITL Waits**](http://blog.itpub.net/26736162/viewspace-1869086/#4300)
* [**Segments by Buffer Busy Waits**](http://blog.itpub.net/26736162/viewspace-1869086/#4400)
* [**Segments by Global Cache Buffer Busy**](http://blog.itpub.net/26736162/viewspace-1869086/#4450)
* [**Segments by CR Blocks Received**](http://blog.itpub.net/26736162/viewspace-1869086/#4500)
* [**Segments by Current Blocks Received**](http://blog.itpub.net/26736162/viewspace-1869086/#4600)

[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

DBA\_HIST\_SEG\_STAT

**desc**DBA\_HIST\_SEG\_STAT

v$sesstat

v$statname

### Segments by Logical Reads

* **Total Logical Reads: 16,648,792**
* **Captured Segments account for 85.2% of Total**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Owner** | **Tablespace Name** | **Object Name** | **Subobject Name** | **Obj. Type** | **Logical Reads** | **%Total** |
| ICCI01 | ICCIDAT01 | ICCICCS\_PK |  | INDEX | 1,544,848 | 9.28 |
| ICCI01 | ICCIDAT01 | CUSCAD\_TMP |  | TABLE | 1,349,536 | 8.11 |
| ICCI01 | ICCIDAT01 | ICCIFNSACT\_PK |  | INDEX | 1,268,400 | 7.62 |
| ICCI01 | ICCIDAT01 | IND\_OLDNEWACT |  | INDEX | 1,071,072 | 6.43 |
| ICCI01 | ICCIDAT01 | CUID\_PK |  | INDEX | 935,584 | 5.62 |

[**Back to Segment Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#30)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Segments by Physical Reads

* **Total Physical Reads: 322,678**
* **Captured Segments account for 64.2% of Total**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Owner** | **Tablespace Name** | **Object Name** | **Subobject Name** | **Obj. Type** | **Physical Reads** | **%Total** |
| ICCI01 | ICCIDAT01 | CUID\_TMP |  | TABLE | 116,417 | 36.08 |
| ICCI01 | ICCIDAT01 | CUMI\_TMP |  | TABLE | 44,086 | 13.66 |
| ICCI01 | ICCIDAT01 | CUSM\_TMP |  | TABLE | 26,078 | 8.08 |
| ICCI01 | ICCIDAT01 | CUSVAA\_TMP\_PK |  | INDEX | 19,554 | 6.06 |
| ICCI01 | ICCIDAT01 | CUID |  | TABLE | 259 | 0.08 |

[**Back to Segment Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#30)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Segments by Row Lock Waits

当一个进程予在正被其它进程锁住的数据行上获得排它锁时发生这种等待。这种等待经常是由于在一个有主键索引的表上做大量INSERT操作。

**No data exists for this section of the report.**

[**Back to Segment Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#30)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)Segments by ITL Waits

**No data exists for this section of the report.**

[**Back to Segment Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#30)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Segments by Buffer Busy Waits

**No data exists for this section of the report.**

[**Back to Segment Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#30)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Segments by Global Cache Buffer Busy

* **% of Capture shows % of GC Buffer Busy for each top segment compared**
* **with GC Buffer Busy for all segments captured by the Snapshot**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Owner** | **Tablespace Name** | **Object Name** | **Subobject Name** | **Obj. Type** | **GC Buffer Busy** | **% of Capture** |
| SYS | SYSTEM | TSQ$ |  | TABLE | 2 | 100.00 |

[**Back to Segment Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#30)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Segments by CR Blocks Received

* **Total CR Blocks Received: 4,142**
* **Captured Segments account for 95.6% of Total**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Owner** | **Tablespace Name** | **Object Name** | **Subobject Name** | **Obj. Type** | **CR Blocks Received** | **%Total** |
| SYS | SYSTEM | USER$ |  | TABLE | 1,001 | 24.17 |
| SYS | SYSTEM | TSQ$ |  | TABLE | 722 | 17.43 |
| SYS | SYSTEM | SEG$ |  | TABLE | 446 | 10.77 |
| SYS | SYSTEM | OBJ$ |  | TABLE | 264 | 6.37 |
| SYS | SYSTEM | I\_OBJ2 |  | INDEX | 174 | 4.20 |

[**Back to Segment Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#30)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Segments by Current Blocks Received

* **Total Current Blocks Received: 15,502**
* **Captured Segments account for 84.8% of Total**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Owner** | **Tablespace Name** | **Object Name** | **Subobject Name** | **Obj. Type** | **Current Blocks Received** | **%Total** |
| ICCI01 | ICCIDAT01 | CUSM\_TMP |  | TABLE | 5,764 | 37.18 |
| ICCI01 | ICCIDAT01 | CUMI\_TMP |  | TABLE | 2,794 | 18.02 |
| ICCI01 | ICCIDAT01 | CUID\_TMP |  | TABLE | 2,585 | 16.68 |
| SYS | SYSTEM | SEG$ |  | TABLE | 361 | 2.33 |
| SYS | SYSTEM | TSQ$ |  | TABLE | 361 | 2.33 |

[**Back to Segment Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#30)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

## Dictionary Cache Statistics

* [**Dictionary Cache Stats**](http://blog.itpub.net/26736162/viewspace-1869086/#4700)
* [**Dictionary Cache Stats (RAC)**](http://blog.itpub.net/26736162/viewspace-1869086/#4800)

[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

**/\* 库缓存详细信息，。**

**Get Requests：get表示一种类型的锁，语法分析锁。这种类型的锁在引用了一个对象的那条SQL语句的语法分析阶段被设置在该对象上。每当一条语句被语法分析一次时 ，Get Requests的值就增加1。**

**pin requests：pin也表示一种类型的锁，是在执行发生的加锁。每当一条语句执行一次，pin requests的值就增加1。**

**reloads：reloads列显示一条已执行过的语句因Library Cache使该语句的已语法分析版本过期或作废而需要被重新语法分析的次数。**

**invalidations：失效发生在一条已告诉缓存的SQL语句即使已经在library cache中，但已被标记为无效并迎词而被迫重新做语法分析的时候。每当已告诉缓存的语句所引用的对象以某种方式被修改时，这些语句就被标记为无效。**

**pct miss应该不高于1％。**

**Reloads /pin requests <1％，否则应该考虑增大SHARED\_POOL\_SIZE。**

**该部分信息通过v$librarycache视图统计得到：**

**select namespace,gethitratio,pinhitratio,reloads,invalidations**

**from v$librarycache**

**where namespace in ('SQL AREA','TABLE/PROCEDURE','BODY','TRIGGER', 'INDEX');**

### Dictionary Cache Stats

* **"Pct Misses" should be very low (< 2% in most cases)**
* **"Final Usage" is the number of cache entries being used**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Cache** | **Get Requests** | **Pct Miss** | **Scan Reqs** | **Pct Miss** | **Mod Reqs** | **Final Usage** |
| dc\_awr\_control | 86 | 0.00 | 0 |  | 4 | 1 |
| dc\_constraints | 59 | 91.53 | 0 |  | 20 | 1,350 |
| dc\_files | 23 | 0.00 | 0 |  | 0 | 23 |
| dc\_global\_oids | 406 | 0.00 | 0 |  | 0 | 35 |
| dc\_histogram\_data | 673 | 0.15 | 0 |  | 0 | 1,555 |
| dc\_histogram\_defs | 472 | 24.36 | 0 |  | 0 | 4,296 |
| dc\_object\_grants | 58 | 0.00 | 0 |  | 0 | 154 |
| dc\_object\_ids | 1,974 | 6.13 | 0 |  | 0 | 1,199 |
| dc\_objects | 955 | 19.58 | 0 |  | 56 | 2,064 |
| dc\_profiles | 30 | 0.00 | 0 |  | 0 | 1 |
| dc\_rollback\_segments | 3,358 | 0.00 | 0 |  | 0 | 37 |
| dc\_segments | 2,770 | 2.56 | 0 |  | 1,579 | 1,312 |
| dc\_sequences | 9 | 33.33 | 0 |  | 9 | 5 |
| dc\_table\_scns | 6 | 100.00 | 0 |  | 0 | 0 |
| dc\_tablespace\_quotas | 1,558 | 28.50 | 0 |  | 1,554 | 3 |
| dc\_tablespaces | 346,651 | 0.00 | 0 |  | 0 | 7 |
| dc\_usernames | 434 | 0.00 | 0 |  | 0 | 14 |
| dc\_users | 175,585 | 0.00 | 0 |  | 0 | 43 |
| outstanding\_alerts | 57 | 71.93 | 0 |  | 0 | 1 |

[**Back to Dictionary Cache Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#31)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Dictionary Cache Stats (RAC)

|  |  |  |  |
| --- | --- | --- | --- |
| **Cache** | **GES Requests** | **GES Conflicts** | **GES Releases** |
| dc\_awr\_control | 8 | 0 | 0 |
| dc\_constraints | 88 | 22 | 0 |
| dc\_histogram\_defs | 115 | 0 | 0 |
| dc\_object\_ids | 143 | 101 | 0 |
| dc\_objects | 253 | 111 | 0 |
| dc\_segments | 3,228 | 49 | 0 |
| dc\_sequences | 17 | 3 | 0 |
| dc\_table\_scns | 6 | 0 | 0 |
| dc\_tablespace\_quotas | 3,093 | 441 | 0 |
| dc\_users | 8 | 1 | 0 |
| outstanding\_alerts | 113 | 41 | 0 |

[**Back to Dictionary Cache Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#31)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

## Library Cache Statistics

* [**Library Cache Activity**](http://blog.itpub.net/26736162/viewspace-1869086/#4900)
* [**Library Cache Activity (RAC)**](http://blog.itpub.net/26736162/viewspace-1869086/#5000)

[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Library Cache Activity

* **"Pct Misses" should be very low**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Namespace** | **Get Requests** | **Pct Miss** | **Pin Requests** | **Pct Miss** | **Reloads** | **Invali- dations** |
| BODY | 105 | 0.00 | 247 | 0.00 | 0 | 0 |
| CLUSTER | 3 | 0.00 | 4 | 0.00 | 0 | 0 |
| INDEX | 13 | 46.15 | 26 | 42.31 | 5 | 0 |
| SQL AREA | 56 | 100.00 | 1,857,002 | 0.02 | 32 | 12 |
| TABLE/PROCEDURE | 179 | 35.75 | 3,477 | 8.02 | 63 | 0 |
| TRIGGER | 323 | 0.00 | 386 | 0.00 | 0 | 0 |

[**Back to Library Cache Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#32)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Library Cache Activity (RAC)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Namespace** | **GES Lock Requests** | **GES Pin Requests** | **GES Pin Releases** | **GES Inval Requests** | **GES Invali- dations** |
| BODY | 5 | 0 | 0 | 0 | 0 |
| CLUSTER | 4 | 0 | 0 | 0 | 0 |
| INDEX | 26 | 22 | 6 | 17 | 0 |
| TABLE/PROCEDURE | 1,949 | 285 | 63 | 244 | 0 |

[**Back to Library Cache Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#32)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

## Memory Statistics

* [**Process Memory Summary**](http://blog.itpub.net/26736162/viewspace-1869086/#7300)
* [**SGA Memory Summary**](http://blog.itpub.net/26736162/viewspace-1869086/#5900)
* [**SGA breakdown difference**](http://blog.itpub.net/26736162/viewspace-1869086/#6100)

[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Process Memory Summary

* **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**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Category** | **Alloc (MB)** | **Used (MB)** | **Avg Alloc (MB)** | **Std Dev Alloc (MB)** | **Max Alloc (MB)** | **Hist Max Alloc (MB)** | **Num Proc** | **Num Alloc** |
| B | Other | 136.42 |  | 5.25 | 8.55 | 24 | 27 | 26 | 26 |
|  | Freeable | 13.50 | 0.00 | 1.50 | 1.11 | 3 |  | 9 | 9 |
|  | SQL | 0.33 | 0.16 | 0.03 | 0.03 | 0 | 2 | 12 | 10 |
|  | PL/SQL | 0.12 | 0.06 | 0.01 | 0.01 | 0 | 0 | 24 | 24 |
| E | Other | 138.65 |  | 4.78 | 8.20 | 24 | 27 | 29 | 29 |
|  | Freeable | 14.94 | 0.00 | 1.36 | 1.04 | 3 |  | 11 | 11 |
|  | SQL | 0.39 | 0.19 | 0.03 | 0.03 | 0 | 2 | 15 | 12 |
|  | PL/SQL | 0.18 | 0.11 | 0.01 | 0.01 | 0 | 0 | 27 | 26 |

[**Back to Memory Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#33)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### SGA Memory Summary

这部分是关于SGA内存分配的一个描述，我们可以通过show sga等命令也可以查看到这里的内容。

Fixed Size:

    oracle 的不同平台和不同版本下可能不一样，但对于确定环境是一个固定的值，里面存储了SGA 各部分组件的信息，可以看作引导建立SGA的区域。

Variable Size:

    包含了shared\_pool\_size、java\_pool\_size、large\_pool\_size 等内存设置。

Database Buffers:

    指数据缓冲区，在8i 中包含db\_block\_buffer\*db\_block\_size、buffer\_pool\_keep、buffer\_pool\_recycle 三部分内存。在9i 中包含db\_cache\_size、db\_keep\_cache\_size、db\_recycle\_cache\_size、 db\_nk\_cache\_size。

Redo Buffers:

    指日志缓冲区，log\_buffer。对于logbuffer，我们会发现在v$parameter、v$sgastat、v$sga的值不一样。v$parameter是我们可以自己设定的值，也可以设定为0，这时候，oracle降会以默认的最小值来设置v$sgastat的值，同时v$sga也是最小的值。v$sgastat的值是基于参数log\_buffer的设定值，再根据一定的计算公式得到的一个值。v$sga的值，则是根据v$sgastat的值，然后选择再加上8k－11k的一个值，得到min(n\*4k)的一个值。就是说得到的结果是4k的整数倍，也就是说v$sga是以4k的单位递增的。

|  |  |  |
| --- | --- | --- |
| **SGA regions** | **Begin Size (Bytes)** | **End Size (Bytes) (if different)** |
| Database Buffers | 3,506,438,144 |  |
| Fixed Size | 2,078,368 |  |
| Redo Buffers | 14,696,448 |  |
| Variable Size | 771,754,336 |  |

[**Back to Memory Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#33)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### SGA breakdown difference

* **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 | 16.00 | 16.00 | 0.00 |
| large | PX msg pool | 1.03 | 1.03 | 0.00 |
| large | free memory | 14.97 | 14.97 | 0.00 |
| shared | ASH buffers | 15.50 | 15.50 | 0.00 |
| shared | CCursor | 8.58 | 8.85 | 3.09 |
| shared | KQR L PO | 8.75 | 8.80 | 0.55 |
| shared | db\_block\_hash\_buckets | 22.50 | 22.50 | 0.00 |
| shared | free memory | 371.80 | 369.61 | -0.59 |
| shared | gcs resources | 66.11 | 66.11 | 0.00 |
| shared | gcs shadows | 41.65 | 41.65 | 0.00 |
| shared | ges big msg buffers | 13.75 | 13.75 | 0.00 |
| shared | ges enqueues | 7.44 | 7.56 | 1.63 |
| shared | ges reserved msg buffers | 7.86 | 7.86 | 0.00 |
| shared | library cache | 10.78 | 10.93 | 1.41 |
| shared | row cache | 7.16 | 7.16 | 0.00 |
| shared | sql area | 27.49 | 28.50 | 3.67 |
|  | buffer\_cache | 3,344.00 | 3,344.00 | 0.00 |
|  | fixed\_sga | 1.98 | 1.98 | 0.00 |
|  | log\_buffer | 14.02 | 14.02 | 0.00 |

[**Back to Memory Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#33)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

## Streams Statistics

* [**Streams CPU/IO Usage**](http://blog.itpub.net/26736162/viewspace-1869086/#6150)
* [**Streams Capture**](http://blog.itpub.net/26736162/viewspace-1869086/#6200)
* [**Streams Apply**](http://blog.itpub.net/26736162/viewspace-1869086/#6300)
* [**Buffered Queues**](http://blog.itpub.net/26736162/viewspace-1869086/#6400)
* [**Buffered Subscribers**](http://blog.itpub.net/26736162/viewspace-1869086/#6500)
* [**Rule Set**](http://blog.itpub.net/26736162/viewspace-1869086/#6600)

[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Streams CPU/IO Usage

**No data exists for this section of the report.**

[**Back to Streams Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#34)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Streams Capture

**No data exists for this section of the report.**

[**Back to Streams Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#34)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Streams Apply

**No data exists for this section of the report.**

[**Back to Streams Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#34)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Buffered Queues

**No data exists for this section of the report.**

[**Back to Streams Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#34)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Buffered Subscribers

**No data exists for this section of the report.**

[**Back to Streams Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#34)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

### Rule Set

**No data exists for this section of the report.**

[**Back to Streams Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#34)[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

## Resource Limit Stats

* **only rows with Current or Maximum Utilization > 80% of Limit are shown**
* **ordered by resource name**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Resource Name** | **Current Utilization** | **Maximum Utilization** | **Initial Allocation** | **Limit** |
| gcs\_resources | 349,392 | 446,903 | 450063 | 450063 |
| gcs\_shadows | 400,300 | 447,369 | 450063 | 450063 |

[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

## init.ora Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter Name** | **Begin value** | **End value (if different)** |
| audit\_file\_dest | /oracle/app/oracle/admin/ICCI/adump |  |
| background\_dump\_dest | /oracle/app/oracle/admin/ICCI/bdump |  |
| cluster\_database | TRUE |  |
| cluster\_database\_instances | 2 |  |
| compatible | 10.2.0.3.0 |  |
| control\_files | /dev/rora\_CTL01, /dev/rora\_CTL02, /dev/rora\_CTL03 |  |
| core\_dump\_dest | /oracle/app/oracle/admin/ICCI/cdump |  |
| db\_block\_size | 8192 |  |
| db\_domain |  |  |
| db\_file\_multiblock\_read\_count | 16 |  |
| db\_name | ICCI |  |
| dispatchers | (PROTOCOL=TCP) (SERVICE=ICCIXDB) |  |
| instance\_number | 1 |  |
| job\_queue\_processes | 10 |  |
| open\_cursors | 800 |  |
| pga\_aggregate\_target | 1073741824 |  |
| processes | 500 |  |
| remote\_listener | LISTENERS\_ICCI |  |
| remote\_login\_passwordfile | EXCLUSIVE |  |
| sga\_max\_size | 4294967296 |  |
| sga\_target | 4294967296 |  |
| sort\_area\_size | 196608 |  |
| spfile | /dev/rora\_SPFILE |  |
| thread | 1 |  |
| undo\_management | AUTO |  |
| undo\_retention | 900 |  |
| undo\_tablespace | UNDOTBS1 |  |
| user\_dump\_dest | /oracle/app/oracle/admin/ICCI/udump |  |

[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

# More RAC Statistics

* [**Global Enqueue Statistics**](http://blog.itpub.net/26736162/viewspace-1869086/#5100)
* [**Global CR Served Stats**](http://blog.itpub.net/26736162/viewspace-1869086/#5200)
* [**Global CURRENT Served Stats**](http://blog.itpub.net/26736162/viewspace-1869086/#5300)
* [**Global Cache Transfer Stats**](http://blog.itpub.net/26736162/viewspace-1869086/#5400)

[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

## Global Enqueue Statistics

|  |  |  |  |
| --- | --- | --- | --- |
| **Statistic** | **Total** | **per Second** | **per Trans** |
| acks for commit broadcast(actual) | 18,537 | 3.92 | 3.31 |
| acks for commit broadcast(logical) | 21,016 | 4.45 | 3.75 |
| broadcast msgs on commit(actual) | 5,193 | 1.10 | 0.93 |
| broadcast msgs on commit(logical) | 5,491 | 1.16 | 0.98 |
| broadcast msgs on commit(wasted) | 450 | 0.10 | 0.08 |
| dynamically allocated gcs resources | 0 | 0.00 | 0.00 |
| dynamically allocated gcs shadows | 0 | 0.00 | 0.00 |
| false posts waiting for scn acks | 0 | 0.00 | 0.00 |
| flow control messages received | 0 | 0.00 | 0.00 |
| flow control messages sent | 2 | 0.00 | 0.00 |
| gcs assume cvt | 0 | 0.00 | 0.00 |
| gcs assume no cvt | 9,675 | 2.05 | 1.73 |
| gcs ast xid | 1 | 0.00 | 0.00 |
| gcs blocked converts | 7,099 | 1.50 | 1.27 |
| gcs blocked cr converts | 8,442 | 1.79 | 1.51 |
| gcs compatible basts | 45 | 0.01 | 0.01 |
| gcs compatible cr basts (global) | 273 | 0.06 | 0.05 |
| gcs compatible cr basts (local) | 12,593 | 2.66 | 2.25 |
| gcs cr basts to PIs | 0 | 0.00 | 0.00 |
| gcs cr serve without current lock | 0 | 0.00 | 0.00 |
| gcs dbwr flush pi msgs | 223 | 0.05 | 0.04 |
| gcs dbwr write request msgs | 223 | 0.05 | 0.04 |
| gcs error msgs | 0 | 0.00 | 0.00 |
| gcs forward cr to pinged instance | 0 | 0.00 | 0.00 |
| gcs immediate (compatible) converts | 2,998 | 0.63 | 0.54 |
| gcs immediate (null) converts | 170,925 | 36.16 | 30.53 |
| gcs immediate cr (compatible) converts | 0 | 0.00 | 0.00 |
| gcs immediate cr (null) converts | 722,748 | 152.88 | 129.11 |
| gcs indirect ast | 306,817 | 64.90 | 54.81 |
| gcs lms flush pi msgs | 0 | 0.00 | 0.00 |
| gcs lms write request msgs | 189 | 0.04 | 0.03 |
| gcs msgs process time(ms) | 16,164 | 3.42 | 2.89 |
| gcs msgs received | 1,792,132 | 379.09 | 320.14 |
| gcs out-of-order msgs | 0 | 0.00 | 0.00 |
| gcs pings refused | 0 | 0.00 | 0.00 |
| gcs pkey conflicts retry | 0 | 0.00 | 0.00 |
| gcs queued converts | 2 | 0.00 | 0.00 |
| gcs recovery claim msgs | 0 | 0.00 | 0.00 |
| gcs refuse xid | 0 | 0.00 | 0.00 |
| gcs regular cr | 0 | 0.00 | 0.00 |
| gcs retry convert request | 0 | 0.00 | 0.00 |
| gcs side channel msgs actual | 437 | 0.09 | 0.08 |
| gcs side channel msgs logical | 21,086 | 4.46 | 3.77 |
| gcs stale cr | 3,300 | 0.70 | 0.59 |
| gcs undo cr | 5 | 0.00 | 0.00 |
| gcs write notification msgs | 23 | 0.00 | 0.00 |
| gcs writes refused | 3 | 0.00 | 0.00 |
| ges msgs process time(ms) | 1,289 | 0.27 | 0.23 |
| ges msgs received | 138,891 | 29.38 | 24.81 |
| global posts dropped | 0 | 0.00 | 0.00 |
| global posts queue time | 0 | 0.00 | 0.00 |
| global posts queued | 0 | 0.00 | 0.00 |
| global posts requested | 0 | 0.00 | 0.00 |
| global posts sent | 0 | 0.00 | 0.00 |
| implicit batch messages received | 81,181 | 17.17 | 14.50 |
| implicit batch messages sent | 19,561 | 4.14 | 3.49 |
| lmd msg send time(ms) | 0 | 0.00 | 0.00 |
| lms(s) msg send time(ms) | 0 | 0.00 | 0.00 |
| messages flow controlled | 15,306 | 3.24 | 2.73 |
| messages queue sent actual | 108,411 | 22.93 | 19.37 |
| messages queue sent logical | 222,518 | 47.07 | 39.75 |
| messages received actual | 474,202 | 100.31 | 84.71 |
| messages received logical | 1,931,144 | 408.50 | 344.97 |
| messages sent directly | 25,742 | 5.45 | 4.60 |
| messages sent indirectly | 137,725 | 29.13 | 24.60 |
| messages sent not implicit batched | 88,859 | 18.80 | 15.87 |
| messages sent pbatched | 1,050,224 | 222.16 | 187.61 |
| msgs causing lmd to send msgs | 61,682 | 13.05 | 11.02 |
| msgs causing lms(s) to send msgs | 85,978 | 18.19 | 15.36 |
| msgs received queue time (ms) | 911,013 | 192.71 | 162.74 |
| msgs received queued | 1,931,121 | 408.50 | 344.97 |
| msgs sent queue time (ms) | 5,651 | 1.20 | 1.01 |
| msgs sent queue time on ksxp (ms) | 66,767 | 14.12 | 11.93 |
| msgs sent queued | 215,124 | 45.51 | 38.43 |
| msgs sent queued on ksxp | 243,729 | 51.56 | 43.54 |
| process batch messages received | 120,003 | 25.38 | 21.44 |
| process batch messages sent | 181,019 | 38.29 | 32.34 |

[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

## Global CR Served Stats

|  |  |
| --- | --- |
| **Statistic** | **Total** |
| CR Block Requests | 10,422 |
| CURRENT Block Requests | 251 |
| Data Block Requests | 10,422 |
| Undo Block Requests | 2 |
| TX Block Requests | 20 |
| Current Results | 10,664 |
| Private results | 4 |
| Zero Results | 5 |
| Disk Read Results | 0 |
| Fail Results | 0 |
| Fairness Down Converts | 1,474 |
| Fairness Clears | 0 |
| Free GC Elements | 0 |
| Flushes | 370 |
| Flushes Queued | 0 |
| Flush Queue Full | 0 |
| Flush Max Time (us) | 0 |
| Light Works | 2 |
| Errors | 0 |

[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

## Global CURRENT Served Stats

* **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 | 17,534 | 99.96 | 0.01 | 0.03 | 0.00 | 0.00 |
| Flushes | 77 | 48.05 | 46.75 | 5.19 | 0.00 | 0.00 |
| Writes | 255 | 5.49 | 53.73 | 40.00 | 0.78 | 0.00 |

[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

## Global Cache Transfer Stats

* **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 No** | **Block Class** | **Blocks Received** | **% Immed** | **% Busy** | **% Congst** | **Blocks Received** | **% Immed** | **% Busy** | **% Congst** |
| 2 | data block | 3,945 | 87.20 | 12.80 | 0.00 | 13,324 | 99.71 | 0.26 | 0.04 |
| 2 | Others | 191 | 100.00 | 0.00 | 0.00 | 2,190 | 96.48 | 3.52 | 0.00 |
| 2 | undo header | 11 | 100.00 | 0.00 | 0.00 | 2 | 100.00 | 0.00 | 0.00 |

[**Back to Top**](http://blog.itpub.net/26736162/viewspace-1869086/#top)

**End of Report**

[[FF1]](http://blog.itpub.net/26736162/viewspace-1869086/#_msoanchor_1)OLAP:联机分析处理

OLTP:联机事务处理

OLAP是主要应用数据仓库系统

OLTP是一般的项目开发用到的基本的、日常的事务处理；比如数据库记录的增、删、改、查。

# 

# 理论实践：循序渐进理解AWR细致入微分析性能报告

黄凯耀 (Kaya) ACOUG核心会员，高级技术专家

曾经工作于Oracle Real World Database Performance Group，一个隶属于Oracle公司总部数据库产品管理的核心团队。大学及研究生时期专注于Linux应用开发和Linux内核开发工作。

编辑手记：AWR是Oracle数据库中一个非常重要的诊断工具，通过度量而展现问题，每一个DBA都应当深入理解这其中的知识，本文通过讲解和分析，展示AWR分析的过程。

概述：本篇文章重点对 AWR 报告中的 DB Time、DBCPU、IO 等数据进行了说明，可帮助读者更加清楚的理解这些数据代表的含义，与数据库的性能表现有何关系。同时通过两个简短的例子，实践如何分析 AWR 报告。

1. AWR 概述

Automatic Workload Repository(AWR) 是10g引入的一个重要组件。在里面存贮着近期一段时间内（默认是7天）数据库活动状态的详细信息。

AWR 报告是对 AWR 视图进行查询而得到的一份自动生成的报告。可以通过下面的脚本手工得到一份 AWR 报告。

通过 AWR 和 AWR 报告，DBA 可以容易地获知最近数据库的活动状态，数据库的各种性能指标的变化趋势曲线，最近数据库可能存在的异常，分析数据库可能存在的性能瓶颈从而对数据库进行优化。

AWR 报告所有的数据来源于 AWR 视图，即以 DBA\_HIST\_开头的所有系统表，Database Reference 有对所有这些系统表的描述，这应该是 Oracle 官方对 AWR 报告的官方注释了。而对于如何有效地去分析 AWR 报告，这可能更需要 DBA 经验的日积月累。

AWR的前身是Statspack，Statspack在10g和11g中也有提供，同时和AWR一起做了同步更新，而且Statspack是公开源代码的，因此，关于Statspack的资料，还有Statspack的源代码，都是理解AWR的一个有用的辅助。

本文着重对AWR中的一些要点进行剖析，欢迎一起讨论AWR相关的问题。

2. DB CPU - CPU负载分析

如果关注数据库的性能，那么当拿到一份 AWR 报告的时候，最想知道的第一件事情可能就是系统资源的利用情况了，而首当其冲的，就是 CPU。

而细分起来，CPU 可能指的是：

OS 级的 User%，Sys%， Idle%

DB 所占 OS CPU 资源的 Busy%

DBCPU 又可以分为前台所消耗的 CPU 和后台所消耗的 CPU

如果数据库的版本是11g，那么很幸运的，这些信息在AWR报告中一目了然：

分析上面的图片，我们可以得出下面的结论：

OS 级的 User%，Sys%，Idle%：

OS 级的 %User 为75.4，%Sys 为2.8，%Idle 为21.2，所以 %Busy 应该是 100-21.1=78.8。

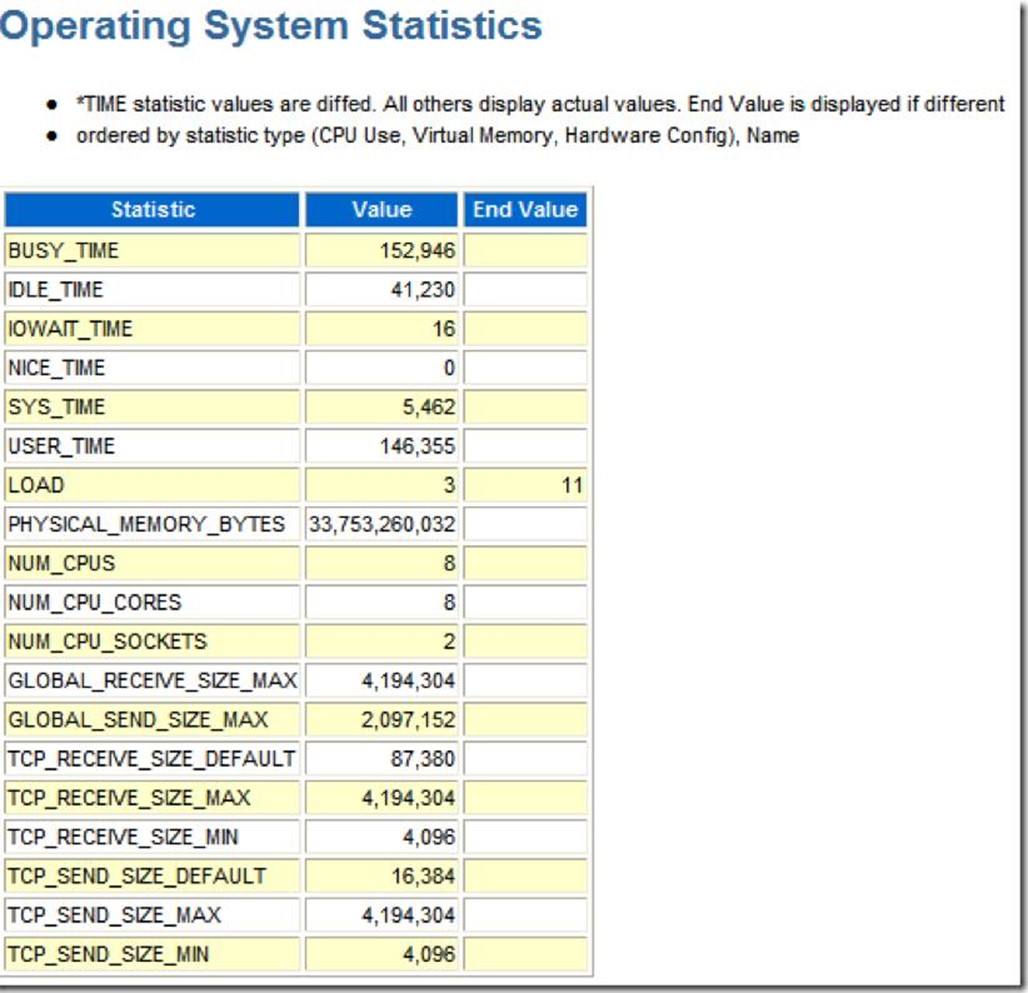
DB所占OSCPU资源的Busy%

DB占了OS CPU资源的69.1，%Busy CPU则可以通过上面的数据得到：

%Busy CPU = %Total CPU/(%Busy) \* 100 = 69.1/78.8 \* 100 = 87.69，和报告的87.7相吻合。

如果是10g呢，则需要手工对 Report 里的一些数据进行计算了。

Host CPU 的结果来源于 DBA\_HIST\_OSSTAT，AWR 报告里已经帮忙整出了这段时间内的绝对数据(这里的时间单位是centi second,也就是1/100秒)



根据上面的数据，稍加计算分析便可得出下面的结果。

OS级的User%，Sys%，Idle%：

%User = USER\_TIME/ (BUSY\_TIME+IDLE\_TIME)\*100 =146355/ (152946+41230)\*100 = 75.37

%Sys = SYS\_TIME/ (BUSY\_TIME+IDLE\_TIME)\*100=5462/(152946+41230)\*100=2.81

%Idle = IDLE\_TIME/ (BUSY\_TIME+IDLE\_TIME)\*100=21230/(152946+41230)\*100=10.93

值得注意的，这里已经隐含着这个AWR报告所捕捉的两个Snapshot之间的时间长短了。有下面的公式：

BUSY\_TIME + IDLE\_TIME = ELAPSED\_TIME \* CPU\_COUNT

**注意：正确的理解这个公式可以对系统CPU资源的使用及其度量的方式有更深一步的理解。**

因此ELAPSED\_TIME =(152946+41230)/8/100 = 242.72 seconds

至于DB对CPU的利用情况，这就涉及到10g新引入的一个关于时间统计的视图V$SYS\_TIME\_MODEL，简单而言，Oracle采用了一个统一的时间模型对一些重要的时间指标进行了记录，具体而言，这些指标包括：

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 (sharingcriteria) elapsed time

5) Hard parse (bindmismatch) elapsed time

3) Failed parse elapsed time

4) Failed parse (out of sharedmemory) 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

这里我们关注的只有和 CPU 相关的两个： Background CPU time 和 DB CPU。

这两个值在 AWR 里面也有记录：



Total DB CPU = DB CPU + Background CPUtime = 1305.89 + 35.91 = 1341.8 seconds

Total DB CPU除以总的 BUSY\_TIME + IDLE\_TIME可得出% Total CPU。

% Total CPU = 1341.8/1941.76 = 69.1%，这刚好与上面Report的值相吻合。

其实，在 Load Profile 部分，我们也可以看出DB对系统CPU的资源利用情况。

用 DBCPU per Second 除以 CPU Count 就可以得到 DB 在前台所消耗的 CPU% 了。这里 5.3/8 = 66.25 %比69.1%稍小，说明DB在后台也消耗了大约3%的 CPU，这是不是一个最简单的方法了呢？

3. DB Time – 进程消耗时间分析

DB CPU 是一个用于衡量 CPU 的使用率的重要指标。假设系统有N个 CPU，那么如果 CPU全部处于繁忙状态的话，一秒钟内的 DBCPU 就是N秒。

如何去表征一个系统的繁忙程度呢？除了利用 CPU 进行计算外，数据库还会利用其它计算资源，如网络、硬盘、内存等等，这些对资源的利用同样可以利用时间进行度量。假设系统有M个 Session 在运行，同一时刻，有的 Session 可能在利用 CPU，有的 Session 可能在访问硬盘，那么，在一秒钟内，所有 Session 的时间加起来就可以表征系统在这一秒内的繁忙程度，一般的，这个和的最大值应该为 M。这其实就是 Oracle 提供的另一个重要指标：DB Time，它用以衡量前端进程所消耗的总时间。

对除 CPU 以外的计算资源的访问，Oracle 用等待事件进行描述。同样地，和 CPU 可分为前台消耗 CPU 和后台消耗 CPU一样，等待事件也可以分为前台等待事件和后台等待事件。

DB Time 一般的应该等于DB CPU + 前台等待事件所消耗时间的总和。等待时间通过 V$SYSTEM\_EVENT 视图进行统计，DB Time 和 DBCPU 则是通过同一个视图，即V$SYS\_TIME\_MODEL 进行统计。

Load Profile 一节就有了对 DB Time 的描述：

这个系统的 CPU 个数是8，因此我们可以知道前台进程用了系统 CPU 的7.1/8=88.75%。 DB Time/s 为11.7，可以看出这个系统是 CPU 非常繁忙的。里面 CPU 占了7.1，则其它前台等待事件占了11.7 –7.1 = 4.6 Wait Time/s。DB CPU 占 DB Time 的比重呢？ 7.1/11.7= 60.68%

Top 5 Timed Events，或许很多人都对它有所耳闻，按照 CPU/ 等待事件占 DB Time 的比例大小，这里列出了 Top 5。如果一个工作负载是 CPU 繁忙型的话，那么在这里应该可以看到 DB CPU 的影子。

注意到，我们刚刚已经算出了 DB CPU 的 %DB time 为60%左右。

其它的 externaltable read，direct path write， PX Deq: read credit， PXDeq: Slave Session Stats 这些就是占比重40%的等待事件里的 Top 4了。

回过头再研究下这个 Top 5 Timed Foreground Events，如果先不看 Load Profile，你能说出这个一个 CPU-Bound 的工作负载吗？

答案是否定的，要知道系统 CPU 的繁忙程序，还要知道这个 AWR 所基于两个 Snapshot 的时间间隔，还要知道系统 CPU 的个数。否则，系统可以是一个很 IDLE 的系统呢。记住 CPU 利用率= DB CPU/(CPU\_COUNT\*Elapsed TIME)。

这个 Top5 给我们的信息只是这个工作负载应该是并行查询，从外部表读取数据，并用 insert append 的方式写入磁盘，同时，主要时间耗费在 CPU 的运算上。

上面提到，DB Time 一般的应该等于 DB CPU + 前台等待事件所消耗时间的总和。在下面有对这三个值的统计：

DB CPU = 6474.65

DB TIME = 10711.2

FG Wait Time = 1182.63

明显的，DBCPU + FG Wait Time < DB Time，只占了71.5%

其它的28.5%被消耗到哪里去了呢？这里其实又隐含着一个 Oracle 如何计算 DBCPU 和 DB Time 的问题。当 CPU 很忙时，如果系统里存在着很多进程，就会发生进程排队等待 CPU 的现象。在这样，DB TIME 是把进程排队等待 CPU 的时间算在内的，而 DB CPU 是不包括这一部分时间。这是造成 DB CPU + FG Wait Time < DB Time 的一个重要原因。如果一个系统 CPU 不忙，这两者应该就比较接近了。

不要忘了在这个例子中，这是一个 CPU 非常繁忙的系统，而71.5%就是一个信号，它提示着这个系统可能是一个 CPU-Bound 的系统。

4. IO数据分析

除了 DBCPU、DB Time，或许另一个比较常用的指标应该是 IO 的利用情况。关于 IO 的指标就比较多了，单单在 Load Profile 里面就有5个，在 DBTime 和 DB CPU 的下面：



这5个指标的值都来自 V$SYSTAT 视图，分别是：

Redo Size: 'redo size'

Logical reads = 'session logical reads'or ('db block gets' + 'consistent gets')

Blocks Changes = 'db block changes'

Physical reads = 'physical reads'

Physical writes = 'physical writes'

具体指标的解释可以参考Database Reference(http://docs.oracle.com/cd/B28359\_01/server.111/b28320/stats002.htm)

如何得到系统大致的 MBPS(Megabits Per Second) 呢？

MBPS= (Physical reads + Physical writes) \*Block\_Size = (196,271.4+2.0)\*8\*1024/1024/1024 = 1533 MB/s

更准确的 MBPS 可以从 Instance Activity Stats 部分获得。

Physical IO disk bytes = physical read total bytes+ physical write total bytes

值得注意的是这里 physical write total bytes 大致是 physical writebytes 的两倍。这应该是 physical write total bytes 统计的是磁盘的 IO，而这里，我们做了 ASM，normal redundancy，一份数据写了两遍的原因。

Load Profile 剩下的部分主要是关于各种执行情况的统计，除了 W/A MB processed 来自 V$PGASTAT（单位其实也是 Byte，不是 MB），其它数据都是来自于 V$SYSSTAT。

Blocks Changes: 'db block changes'

User calls: 'user calls'

Parses: 'parse count (total)'

Hard parses: 'parse count (hard)'

Logons: 'logons cumulative'

Executes: 'execute count'

Rollbacks: 'user rollbacks'

Tranasactions: 'user rollbacks' + 'usercommits'

W/A MB processed: 'bytes processed'

一般而言，Hard parses < Parses < Executes < User Calls。

AWR 的一般性介绍我想差不多就这些了，其它部分的介绍借助于一些更具体的 AWR 报告进行分析可能会更加方便和清晰。

5. AWR 报告分析 – 实战1

构建 DSS 系统的第一步离不开数据加载，通过文本文件加载是最常见的方式，Oracle 提供了外部表加载的方法，即把一个文本文件当成一个正常的表来进行操作，通过类似 insert /\*+ append \*/ into table select from external\_table 的方式进行加载。

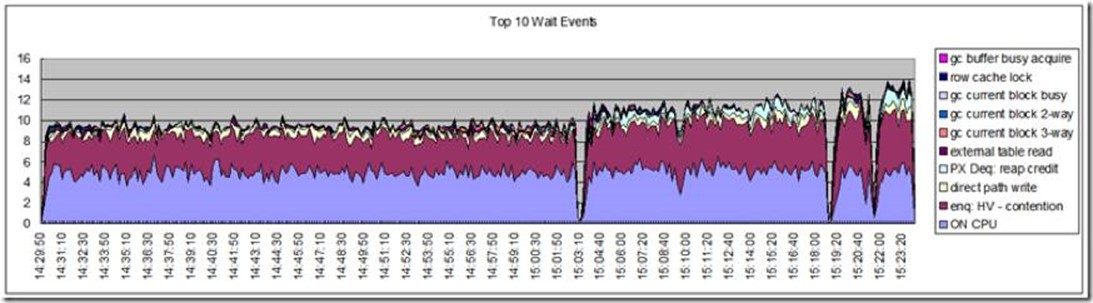
数据加载是一个 CPU-Bound 的过程，不过是通过什么工具，external table 也好，sqlldr 也好，imp 也好，impdp 也好。换句话说，如果连数据加载都出现 IO 瓶颈，这个系统的配置就说不过去了。

这个过程的 AWR 报告会是什么样子的呢？

先做个一般的假定，从外部表加载数据到一个本地分区表。

Top 5 TimedEvents 类似下面：

如果去抓取这段时间 DBA\_HIST\_ACTIVE\_SESS\_HISTORY 的数据，并转换为图表的话，我们会得到更形象的 Top 10 Wait Events。（如何实现这一步可以参考用 Oracle 实现 ASH 的数据透视图：http://www.cnblogs.com/rootq/archive/2009/09/24/1573200.html）



enq: HV –contention 是什么东西呢？

在11.2以前，对于分区表的 parallel direct-path load，Oracle 采用的是 brokered load 的方式，即所有的 PX Slaves 共享对每个分区的 high water mark 的访问，通过轮流持有 high water mark 实现对每个 segment 添加新的 blocks。这种方法对于充分利用 extent 的空间是有帮助的，不过带来的问题就是对 high water mark 的竞争，也就是这里的 enq: HV – contention。在执行计划中，这以 RANDOM LOCAL 标记。下面是一个例子：

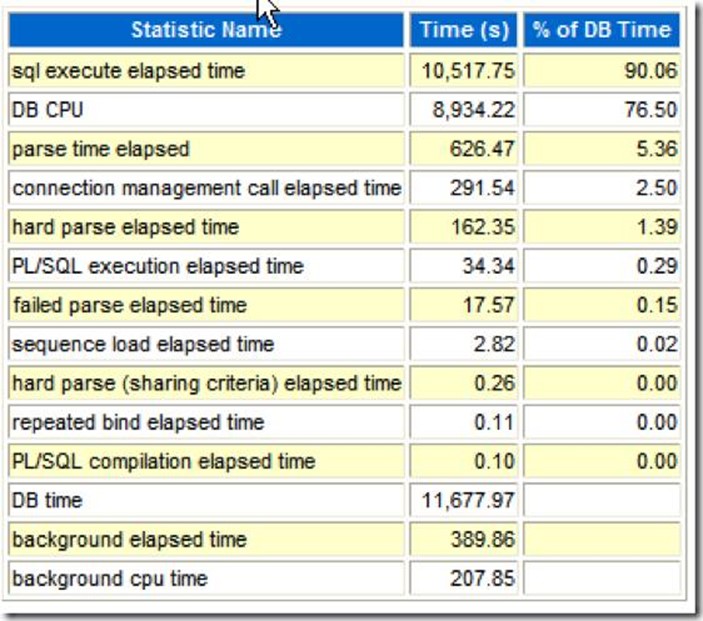
一个好消息是，11.2引入了一种新的方式，叫做 PKEY distribution。在这种方式下，一个特定的分区只交给一个或多个特定的 PX slave 负责，这种方式不仅减少了对 high water mark 的争用，而且可以实现 Partition 内更好的压缩率。

6. AWR报告分析 – 实战2

有一次跟一个 QQ 上的朋友一起探讨了另一个对系统 CPU 进行度量的指标： CPUused by this session。

他刚好有一份 AWR 报告，在这份报告里，出现了严重的 CPU used by this session 和 DB CPU 不一致的现象。

下面是这份报告的一些片断：



再做进一步的归纳：

OS Busy% =1821080/(1821080+5384293) = 25%

Inst CPU% (usingDB CPU) = 8934.22\*100/ (1821080+5384293)=12%

Inst CPU% (usingCPU used by this session) = 418035/ (1821080+5384293) = 6%

用 CPU used by this session 计算出的 CPU 利用率竟然只是用 DB CPU 计算出来的利用通率的一半！

我的第一个反应是在 Jonathan Lewis 网站看到的一篇相关文章，里面提到了 DB CPU 和 CPUused by this session 计算时的不同之处：(https://jonathanlewis.wordpress.com/2009/05/26/cpu-used/)

"Prior to10g Oracle usually updated time figures at the end of each database call; butfrom 10g there are some views where time is updated more regularly.

The "DB CPU" from v$sess\_time\_model increases every six seconds, while the "CPU used by this session" from v$sesstat changes only at the end of the test."

如何验证这一点呢？

在浏览这份报告的 TOP SQL 时，我们发现了下面的现象：

这是从 SQL ordered by Elapsed Time 截取出来的 Top 3 SQL。TOP 1 的 SQL 用了 DB Time 的30.10%，用了2517s 的 CPU Time。但请注意它的 Executions 的值却为0。也就是说，这里的 CPU Time 是还没有被计算入 CPU used by this session 这个指标里面的。

我们再把2517s加回来，看出误差缩小多少：

(251700+418035)/(1821080+5384293) = 9%

这时和用 DB CPU 计算出来的12%还是有1/4的差距。

从这个例子可以看出，用 DB CPU 度量还是比用 CPU usedby this session 来得准确的。特别在有大查询在跑的过程中抓的 AWR，这个误差很有可能会被放大。这是一个有趣的实际例子。

7. 总结

AWR 是分析数据库运行状况的利器，将其运用好可帮助 DBA 提早发现数据库中存在的问题并加以解决。文中主要对 DB CPU、DB Time、IO 等 AWR 报告中的数据进行了分析说明，当然分析AWR报告不能仅限于此，更需要DBA日积月累的经验。希望本文对想了解 AWR 的朋友有一定帮助。

|  |  |  |
| --- | --- | --- |
| **常见问题:如何使用AWR报告来诊断数据库性能问题 (文档 ID 1523048.1)** | [[转到底部](https://support.oracle.com/epmos/faces/DocumentDisplay?_afrLoop=89081458011252&id=1523048.1&_adf.ctrl-state=19ic6slzwj_223)转到底部](https://support.oracle.com/epmos/faces/DocumentDisplay?_afrLoop=89081458011252&id=1523048.1&_adf.ctrl-state=19ic6slzwj_223) | https://support.oracle.com/epmos/adf/images/t.gif |

https://support.oracle.com/epmos/adf/images/t.gif

**文档内容**

|  |  |
| --- | --- |
|  | [目标](https://support.oracle.com/epmos/faces/DocumentDisplay?_afrLoop=89081458011252&id=1523048.1&_adf.ctrl-state=19ic6slzwj_223#GOAL) |
|  | [最佳实践](https://support.oracle.com/epmos/faces/DocumentDisplay?_afrLoop=89081458011252&id=1523048.1&_adf.ctrl-state=19ic6slzwj_223#aref_section11) |

|  |  |  |
| --- | --- | --- |
|  | | [如何主动避免问题发生及做好诊断信息的收集](https://support.oracle.com/epmos/faces/DocumentDisplay?_afrLoop=89081458011252&id=1523048.1&_adf.ctrl-state=19ic6slzwj_223#aref_section12) |
|  | [提出问题、获取帮助并分享您的经验](https://support.oracle.com/epmos/faces/DocumentDisplay?_afrLoop=89081458011252&id=1523048.1&_adf.ctrl-state=19ic6slzwj_223#aref_section13) | |

|  |  |
| --- | --- |
|  | [解决方案](https://support.oracle.com/epmos/faces/DocumentDisplay?_afrLoop=89081458011252&id=1523048.1&_adf.ctrl-state=19ic6slzwj_223#FIX) |
|  | [Interpretation](https://support.oracle.com/epmos/faces/DocumentDisplay?_afrLoop=89081458011252&id=1523048.1&_adf.ctrl-state=19ic6slzwj_223#aref_section21) |

|  |  |
| --- | --- |
|  | [Top 5 Timed Events](https://support.oracle.com/epmos/faces/DocumentDisplay?_afrLoop=89081458011252&id=1523048.1&_adf.ctrl-state=19ic6slzwj_223#aref_section22) |
|  | [SQL Statistics](https://support.oracle.com/epmos/faces/DocumentDisplay?_afrLoop=89081458011252&id=1523048.1&_adf.ctrl-state=19ic6slzwj_223#aref_section23) |

|  |  |
| --- | --- |
|  | [分析：](https://support.oracle.com/epmos/faces/DocumentDisplay?_afrLoop=89081458011252&id=1523048.1&_adf.ctrl-state=19ic6slzwj_223#aref_section24) |
|  | [Other SQL Statistic Sections](https://support.oracle.com/epmos/faces/DocumentDisplay?_afrLoop=89081458011252&id=1523048.1&_adf.ctrl-state=19ic6slzwj_223#aref_section25) |

|  |  |  |
| --- | --- | --- |
|  | | [Waits for 'Cursor: mutex/pin'](https://support.oracle.com/epmos/faces/DocumentDisplay?_afrLoop=89081458011252&id=1523048.1&_adf.ctrl-state=19ic6slzwj_223#aref_section26) |
|  | [Load Profile](https://support.oracle.com/epmos/faces/DocumentDisplay?_afrLoop=89081458011252&id=1523048.1&_adf.ctrl-state=19ic6slzwj_223#aref_section27) | |

|  |  |
| --- | --- |
|  | [Instance Efficiency](https://support.oracle.com/epmos/faces/DocumentDisplay?_afrLoop=89081458011252&id=1523048.1&_adf.ctrl-state=19ic6slzwj_223#aref_section28) |
|  | [Latch Activity](https://support.oracle.com/epmos/faces/DocumentDisplay?_afrLoop=89081458011252&id=1523048.1&_adf.ctrl-state=19ic6slzwj_223#aref_section29) |

|  |  |  |
| --- | --- | --- |
|  | [值得注意的wait events](https://support.oracle.com/epmos/faces/DocumentDisplay?_afrLoop=89081458011252&id=1523048.1&_adf.ctrl-state=19ic6slzwj_223#aref_section210) | |
|  | | [CPU time events](https://support.oracle.com/epmos/faces/DocumentDisplay?_afrLoop=89081458011252&id=1523048.1&_adf.ctrl-state=19ic6slzwj_223#aref_section211) |

|  |  |
| --- | --- |
|  | [Analysis:](https://support.oracle.com/epmos/faces/DocumentDisplay?_afrLoop=89081458011252&id=1523048.1&_adf.ctrl-state=19ic6slzwj_223#aref_section212) |
|  | [其他潜在的CPU相关的问题：](https://support.oracle.com/epmos/faces/DocumentDisplay?_afrLoop=89081458011252&id=1523048.1&_adf.ctrl-state=19ic6slzwj_223#aref_section213) |

|  |  |
| --- | --- |
|  | [检查是否有其他等待事件与高CPU 事件同时出现](https://support.oracle.com/epmos/faces/DocumentDisplay?_afrLoop=89081458011252&id=1523048.1&_adf.ctrl-state=19ic6slzwj_223#aref_section214) |
|  | [数据库以外的CPU使用率过高](https://support.oracle.com/epmos/faces/DocumentDisplay?_afrLoop=89081458011252&id=1523048.1&_adf.ctrl-state=19ic6slzwj_223#aref_section215) |

|  |  |  |
| --- | --- | --- |
|  | | [诊断CPU使用率](https://support.oracle.com/epmos/faces/DocumentDisplay?_afrLoop=89081458011252&id=1523048.1&_adf.ctrl-state=19ic6slzwj_223#aref_section216) |
|  | ['Log file sync' waits](https://support.oracle.com/epmos/faces/DocumentDisplay?_afrLoop=89081458011252&id=1523048.1&_adf.ctrl-state=19ic6slzwj_223#aref_section217) | |

|  |  |  |
| --- | --- | --- |
|  | | [Buffer busy waits](https://support.oracle.com/epmos/faces/DocumentDisplay?_afrLoop=89081458011252&id=1523048.1&_adf.ctrl-state=19ic6slzwj_223#aref_section218) |
|  | [诊断其他问题](https://support.oracle.com/epmos/faces/DocumentDisplay?_afrLoop=89081458011252&id=1523048.1&_adf.ctrl-state=19ic6slzwj_223#aref_section219) | |

|  |  |
| --- | --- |
|  | [使用ADDM的报告](https://support.oracle.com/epmos/faces/DocumentDisplay?_afrLoop=89081458011252&id=1523048.1&_adf.ctrl-state=19ic6slzwj_223#aref_section220) |
|  | [其他的AWR参考文章](https://support.oracle.com/epmos/faces/DocumentDisplay?_afrLoop=89081458011252&id=1523048.1&_adf.ctrl-state=19ic6slzwj_223#aref_section221) |

|  |  |
| --- | --- |
|  | [Statspack](https://support.oracle.com/epmos/faces/DocumentDisplay?_afrLoop=89081458011252&id=1523048.1&_adf.ctrl-state=19ic6slzwj_223#aref_section222) |
|  | [参考](https://support.oracle.com/epmos/faces/DocumentDisplay?_afrLoop=89081458011252&id=1523048.1&_adf.ctrl-state=19ic6slzwj_223#REF) |

## 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](https://www.baidu.com/s?wd=%E6%81%AF%E6%81%AF%E7%9B%B8%E5%85%B3&tn=24004469_oem_dg&rsv_dl=gh_pc_csdn)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](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=1482811.1) Best Practices: Proactively Avoiding Database and Query Performance Issues  
[Document 1477599.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=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](https://community.oracle.com/message/12160644) 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](https://community.oracle.com/community/support/oracle_database/database_tuning)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](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=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](https://support.oracle.com/epmos/faces/DocumentDisplay?_afrLoop=89081458011252&id=1523048.1&_adf.ctrl-state=19ic6slzwj_223#ADDM)

### 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](https://support.oracle.com/epmos/faces/DocumentDisplay?_afrLoop=89081458011252&id=1523048.1&_adf.ctrl-state=19ic6slzwj_223#SQLStats)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](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=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](https://support.oracle.com/epmos/faces/DocumentDisplay?_afrLoop=89081458011252&id=1523048.1&_adf.ctrl-state=19ic6slzwj_223#SQLStats) 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：  
  
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](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=271196.1) Automatic SQL Tuning - SQL Profiles.  
  
[Document 262687.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=262687.1) How to use the Sql Tuning Advisor.  
[Document 276103.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=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](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=1356828.1) FAQ: 'cursor: mutex ..' / 'cursor: pin ..' / 'library cache: mutex ..' Type Wait Events   
[Note:1349387.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=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](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=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](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=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](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=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](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=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](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=1376916.1) Troubleshooting: "Log File Sync" Waits   
[Note:34592.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=34592.1)WAITEVENT: "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](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=155971.1) Resolving Intense and "Random" Buffer Busy Wait Performance Problems:[Note:34405.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=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](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=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:250655.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=250655.1)How 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](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=786554.1) How to Read PGA Memory Advisory Section in AWR and Statspack Reports  
[Document 754639.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=754639.1) How to Read Buffer Cache Advisory Section in AWR and Statspack Reports  
  
[Document 1301503.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=1301503.1) Troubleshooting: AWR Snapshot Collection issues   
[Document 1363422.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=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](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=94224.1) FAQ- Statspack Complete Reference  
[Document  394937.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=394937.1) Statistics Package (STATSPACK) Guide   
  
[Document 149113.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=149113.1)  Installing and Configuring StatsPack Package  
[Document 149121.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=149121.1) Gathering a StatsPack snapshot  
[Document 228913.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=228913.1)  Systemwide Tuning using STATSPACK Reports

## 参考

[NOTE:1349387.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=1349387.1) - Troubleshooting 'cursor: pin S wait on X' waits.  
[NOTE:6904068.8](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=6904068.8) - Bug 6904068 - High CPU usage when there are "cursor: pin S" waits  
[NOTE:262687.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=262687.1) - Using the DBMS\_SQLTUNE package to Run the Sql Tuning Advisor  
[NOTE:271196.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=271196.1) - Automatic SQL Tuning and SQL Profiles  
[NOTE:276103.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=276103.1) - Performance Tuning Using Advisors and Manageability Features: AWR, ASH, ADDM and Sql Tuning Advisor  
[NOTE:34405.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=34405.1) - WAITEVENT: "buffer busy waits" Reference Note  
[NOTE:250655.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=250655.1) - How to use the Automatic Database Diagnostic Monitor  
[NOTE:223117.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=223117.1) - Troubleshooting I/O Related Waits  
[NOTE:1482811.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=1482811.1) - Best Practices: Proactively Avoiding Database and Query Performance Issues  
[NOTE:1477599.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=1477599.1) - Best Practices: Proactive Data Collection for Performance Issues  
[NOTE:34592.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=34592.1) - WAITEVENT: "log file sync" Reference Note  
[NOTE:413942.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=413942.1) - How to Identify Which Latch is Associated with a "latch free" wait  
[NOTE:433472.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=433472.1) - OS Watcher For Windows (OSWFW) User Guide  
[NOTE:1356828.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=1356828.1) - FAQ: 'cursor: mutex ..' / 'cursor: pin ..' / 'library cache: mutex ..' Type Wait Events  
[NOTE:1363422.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=1363422.1) - Automatic Workload Repository (AWR) Reports - Main Information Sources  
[NOTE:1377446.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=1377446.1) - \* Troubleshooting Performance Issues  
[NOTE:1376916.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=1376916.1) - Troubleshooting: 'Log file sync' Waits  
[NOTE:155971.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=155971.1) - Resolving Intense and "Random" Buffer Busy Wait Performance Problems  
[NOTE:164768.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=164768.1) - Troubleshooting: High CPU Utilization  
[NOTE:228913.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=228913.1) - Systemwide Tuning using STATSPACK Reports  
[NOTE:754639.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=754639.1) - How to Read Buffer Cache Advisory Section in AWR and Statspack Reports.  
[NOTE:786554.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=786554.1) - How to Read PGA Memory Advisory Section in AWR and Statspack Reports to Tune PGA\_AGGREGATE\_TARGET  
[BUG:6904068](https://support.oracle.com/epmos/faces/BugDisplay?parent=DOCUMENT&sourceId=1523048.1&id=6904068) - HIGH CPU UTILIZATION DURING MUTEX WAIT EVENTS  
[NOTE:1301503.1](https://support.oracle.com/epmos/faces/DocumentDisplay?parent=DOCUMENT&sourceId=1523048.1&id=1301503.1) - Troubleshooting: AWR Snapshot Collection Issues