**Oracle AWR报告解读**

**2019年08月19日**

# 、Oracle AWR报告简介

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。

# 、Oracle AWR报告生成

## AWR生成脚本

脚本目录 $ORACLE\_HOME/rdbms/admin

@?/rdbms/admin/addmrpt.sql  
@?/rdbms/admin/awrrpt.sql  
@?/rdbms/admin/ashrpt.sql  
@?/rdbms/admin/awrsqrpt.sql

## 快照、基线

-- 查看快照保留期限，11g默认为8天  
SELECT retention FROM dba\_hist\_wr\_control;  
-- 修改快照时间间隔  
EXEC DBMS\_WORKLOAD\_REPOSITORY.MODIFY\_SNAPSHOT\_SETTINGS( interval => 30);  
-- 手动生成快照  
EXEC DBMS\_WORKLOAD\_REPOSITORY.CREATE\_SNAPSHOT('TYPICAL');  
或  
BEGIN   
DBMS\_WORKLOAD\_REPOSITORY.create\_snapshot();   
END;   
/  
-- 生成 AWR 基线：  
BEGIN   
DBMS\_WORKLOAD\_REPOSITORY.create\_baseline (   
  start\_snap\_id => 10,    
  end\_snap\_id   => 100,   
  baseline\_name => 'AWR First baseline');   
END;   
/  
-- 生成 AWR 基线(11g)：  
BEGIN  
DBMS\_WORKLOAD\_REPOSITORY.CREATE\_BASELINE\_TEMPLATE (  
start\_time => to\_date('&start\_date\_time','&start\_date\_time\_format'),  
end\_time => to\_date('&end\_date\_time','&end\_date\_time\_format'),  
baseline\_name => 'MORNING',  
template\_name => 'MORNING',  
expiration => NULL ) ;  
END;  
/  
-- 基于重复时间周期来制定用于创建和删除 AWR 基线的模板：  
BEGIN  
DBMS\_WORKLOAD\_REPOSITORY.CREATE\_BASELINE\_TEMPLATE (  
day\_of\_week => 'MONDAY',  
hour\_in\_day => 9,  
duration => 3,  
start\_time => to\_date('&start\_date\_time','&start\_date\_time\_format'),  
end\_time => to\_date('&end\_date\_time','&end\_date\_time\_format'),  
baseline\_name\_prefix => 'MONDAY\_MORNING'  
template\_name => 'MONDAY\_MORNING',  
expiration => 30 );  
END;  
/  
  
-- 删除 AWR 基线：  
BEGIN  
  DBMS\_WORKLOAD\_REPOSITORY.DROP\_BASELINE (baseline\_name => 'AWR First baseline');  
END;  
/

## 其他脚本

awrrpt.sql   
展示一段时间范围两个快照之间的数据库性能指标。  
awrrpti.sql   
展示一段时间范围两个快照之间的特定数据库和特定实例的性能指标。  
awrsqrpt.sql  
展示特定 SQL 在一段时间范围两个快照之间的性能指标，运行这个脚本来检查和诊断一个特定 SQL 的性能问题。  
awrsqrpi.sql   
展示特定 SQL 在特定数据库和特定实例的一段时间范围内两个快照之间的性能指标。  
awrddrpt.sql  
用于比较两个指定的时间段之间数据库详细性能指标和配置情况。  
awrddrpi.sql   
用于在特定的数据库和特定实例上，比较两个指定的时间段之间的数据库详细性能指标和配置情况。

## AWR 相关的视图

如下系统视图与 AWR 相关：

V$ACTIVE\_SESSION\_HISTORY - 展示每秒采样的 active session history (ASH)。  
V$METRIC - 展示度量信息。  
V$METRICNAME - 展示每个度量组的度量信息。  
V$METRIC\_HISTORY - 展示历史度量信息。  
V$METRICGROUP - 展示所有的度量组。  
DBA\_HIST\_ACTIVE\_SESS\_HISTORY - 展示 active session history 的历史信息。  
DBA\_HIST\_BASELINE - 展示 AWR 基线信息。  
DBA\_HIST\_DATABASE\_INSTANCE - 展示数据库环境信息。  
DBA\_HIST\_SNAPSHOT - 展示 AWR 快照信息。  
DBA\_HIST\_SQL\_PLAN - 展示 SQL 执行计划信息。  
DBA\_HIST\_WR\_CONTROL - 展示 AWR 设置信息。

# 、Main Report

**Main Report**

* **Wait Events Statistics**
* **SQL Statistics**
* **Instance Activity Statistics**
* **IO Stats**
* **Buffer Pool Statistics**
* **Advisory Statistics**
* **Wait Statistics**
* **Undo Statistics**
* **Latch Statistics**
* **Segment Statistics**
* **Dictionary Cache Statistics**
* **Library Cache Statistics**
* **Memory Statistics**
* **Streams Statistics**
* **Resource Limit Statistics**
* **init.ora Parameters**

# 、Report Summary

## DB Time

**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，2 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。

## Cache Size

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **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。library 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 UndoBlock 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

**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 OLTPT 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 / parsetime 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 |

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在等待上。

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。

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。

# Wait Events Statistics

**Wait Events Statistics**

* **Time Model Statistics**
* **Wait Class**
* **Wait Events**
* **Background Wait Events**
* **Operating System Statistics**
* **Service Statistics**
* **Service Wait Class Stats**

/\* 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**。

## 常见的等待事件

对于常见的等待事件，说明如下：

### 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。

### 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。

### 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：

* 1. 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。

* 1. 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。
  2. 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。
  3. 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。

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

Oracle 10g

block# id

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';

1. 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

* 1. 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
   1. 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

* 1. 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

### 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。

### 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**

### 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。

### 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。

### 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。

### 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。

### 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。

### 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。

### 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。

### 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。

### 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）。

### 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。

### 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。

### 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。

### 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。

### 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。

### 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。

### 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。

## 常见的IDLE wait 事件

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

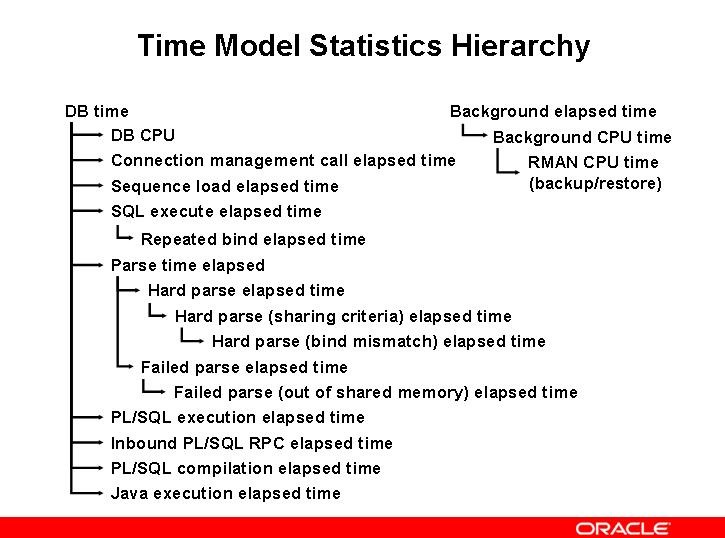
SQL\*Net message from client

## 常见的等待事件和解决方法

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

## 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 |

## 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)**

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;

1. 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。

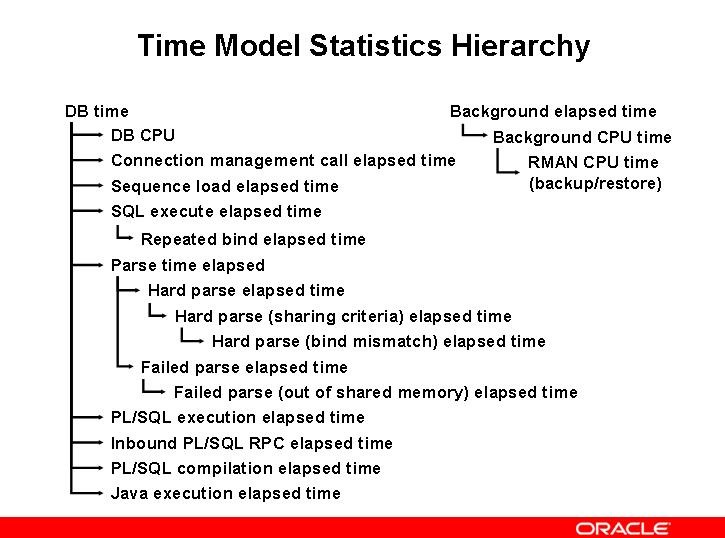
**Log File Parallel Write** 事件是在等待 LGWR 进程将 REDO 记录从 LOG 缓冲区写到联机日志文件时发生的。虽然写操作可能是并发的，但 LGWR 需要等待最后的 I/O 写到磁盘上才能认为并行写的完成，因此等待时间依赖于 OS 完成所有请求的时间。如果这个等待比较严重，可以通过将 LOG 文件移到更快的磁盘上或者条带化磁盘（减少争用）而降低这个等待。

**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）。

## 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 | |

## 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 sarIOWAIT\_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）

## 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 |

## 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 |

# SQL Statistics

* + **SQL ordered by Elapsed Time**
  + **SQL ordered by CPU Time**
  + **SQL ordered by Gets**
  + **SQL ordered by Reads**
  + **SQL ordered by Executions**
  + **SQL ordered by Parse Calls**
  + **SQL ordered by Sharable Memory**
  + **SQL ordered by Version Count**
  + **SQL ordered by Cluster Wait Time**
  + **Complete List of SQL Text**

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;

## 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**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Elapse d Time (s)** | | **CPU**  **Tim e (s)** | **Execution s** | **Elap per Exe c (s)** | **%**  **Tota l DB Tim e** | **SQL Id** | **SQL Module** | **SQL Text** |
| 93 | | 57 | 1 | 93.50 | 14.10 | **d8z0u8hgj8xdy** | cuidmain@HPGICCI1  (TNS V1-V3) | insert into CUID select  CUID\_... |
| 76 | | 75 | 172,329 | 0.00 | 11.52 | **4vja2k2gdtyup** | load\_fnsact@HPGICCI1  (TNS V1-V3) | insert into ICCICCS  values (:... |
| 58 | | 42 | 1 | 58.04 | 8.75 | **569r5k05drsj7** | cumimain@HPGICCI1  (TNS V1-V3) | insert into CUMI select  CUSV\_... |
| 51 | | 42 | 1 | 50.93 | 7.68 | **ackxqhnktxnbc** | cusmmain@HPGICCI1  (TNS V1-V3) | insert into CUSM select  CUSM\_... |
| 38 | | 36 | 166,069 | 0.00 | 5.67 | **7gtztzv329wg0** |  | select c.name, u.name  from co... |
| 35 | | 3 | 1 | 35.00 | 5.28 | **6z06gcfw39pk d** | SQL\*Plus | SELECT  F.TABLESPACE\_NAME, TO\_... |
| 23 | | 23 | 172,329 | 0.00 | 3.46 | **1dm3bq36vu3g**  **8** | load\_fnsact@HPGICCI1  (TNS V1-V3) | insert into iccifnsact  values... |
| 15 | | 11 | 5 | 2.98 | 2.25 | **djs2w2f17nw2z** |  | DECLARE job  BINARY\_INTEGER := ... |
| 14 | | 14 | 172,983 | 0.00 | 2.16 | **7wwv1ybs9zgu**  **z** | load\_fnsact@HPGICCI1  (TNS V1-V3) | update ICCIFNSACT set  BORM\_AD... |
| 13 | | 13 | 172,337 | 0.00 | 2.00 | **gmn2w09rdxn1**  **4** | load\_oldnewact@HPGIC  CI1 (TNS V1-V3) | insert into OLDNEWACT  values ... |
| 13 | | 13 | 166,051 | 0.00 | 1.89 | **chjmy0dxf9mbj** | icci\_migact@HPGICCI1  (TNS V1-V3) | insert into ICCICCS  values (:... |
| 10 | 4 | | 1 | 9.70 | 1.46 | **0yv9t4qb1zb2b** | cuidmain@HPGICCI1  (TNS V1-V3) | select CUID\_CUST\_NO ,  CUID\_ID\_... |
|  |
| 10 | 8 | | 5 | 1.91 | 1.44 | **1crajpb7j5tyz** |  | INSERT INTO STATS$SGA\_TARGET\_A  ... |
| 8 | 8 | | 172,329 | 0.00 | 1.25 | **38apjgr0p55ns** | load\_fnsact@HPGICCI1  (TNS V1-V3) | update ICCICCS set  CCSMAXOVER... |
|  |
| 8 | 8 | | 172,983 | 0.00 | 1.16 | **5c4qu2zmj3gu**  **x** | load\_fnsact@HPGICCI1  (TNS V1-V3) | select \* from  ICCIPRODCODE wh... |

## 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**  **Tim e (s)** | | **Elapse d Time (s)** | | **Execution s** | | **CPU**  **per Exe c (s)** | | **%**  **Tota l DB Tim e** | | **SQL Id** | | | **SQL Module** | | **SQL Text** |
| 75 | | 76 | | 172,329 | | 0.00 | | 11.52 | | **4vja2k2gdtyup** | | | load\_fnsact@HPGICCI1  (TNS V1-V3) | | insert into ICCICCS  values (:... |
| 57 | | 93 | | 1 | | 57.31 | | 14.10 | | **d8z0u8hgj8xdy** | | | cuidmain@HPGICCI1  (TNS V1-V3) | | insert into CUID select  CUID\_... |
|  | | |
| 42 | | 51 | | 1 | | 42.43 | | 7.68 | | **ackxqhnktxnbc** | | | cusmmain@HPGICCI1  (TNS V1-V3) | | insert into CUSM select  CUSM\_... |
| 42 | | 58 | | 1 | | 42.01 | | 8.75 | | **569r5k05drsj7** | | | cumimain@HPGICCI1  (TNS V1-V3) | | insert into CUMI select  CUSV\_... |
|  |  | |
| 36 | | 38 | | 166,069 | | 0.00 | | 5.67 | | **7gtztzv329wg0** | | |  | | select c.name, u.name  from co... |
| 23 | | 23 | | 172,329 | | 0.00 | | 3.46 | | **1dm3bq36vu3g** | | | load\_fnsact@HPGICCI1  (TNS V1-V3) | | insert into iccifnsact  values... |
| **8** | | |
| 14 | | 14 | | 172,983 | | 0.00 | | 2.16 | | **7wwv1ybs9zgu**  **z** | | | load\_fnsact@HPGICCI1  (TNS V1-V3) | | update ICCIFNSACT set  BORM\_AD... |
| 13 | | 13 | | 172,337 | | 0.00 | | 2.00 | | **gmn2w09rdxn1** | | | load\_oldnewact@HPGIC  CI1 (TNS V1-V3) | | insert into OLDNEWACT  values ... |
| **4** | | |
| 13 | | 13 | | 166,051 | | 0.00 | | 1.89 | | **chjmy0dxf9mbj** | | | icci\_migact@HPGICCI1 | | insert into ICCICCS |
|  | |  | |  | |  | |  | |  | | | (TNS V1-V3) | | values (:... | |
| 11 | | 15 | | 5 | | 2.23 | | 2.25 | | **djs2w2f17nw2z** | | |  | | DECLARE job  BINARY\_INTEGER := ... | |
| 8 | | 8 | | 172,329 | | 0.00 | | 1.25 | | **38apjgr0p55ns** | | | load\_fnsact@HPGICCI1  (TNS V1-V3) | | update ICCICCS set  CCSMAXOVER... | |
| 8 | | 10 | | 5 | | 1.60 | | 1.44 | | **1crajpb7j5tyz** | | |  | | INSERT INTO STATS$SGA\_TARGET\_A  ... | |
| 8 | | 8 | | 172,983 | | 0.00 | | 1.16 | | **5c4qu2zmj3gu**  **x** | | | load\_fnsact@HPGICCI1  (TNS V1-V3) | | select \* from  ICCIPRODCODE wh... | |
| 4 | | 10 | | 1 | | 3.54 | | 1.46 | | **0yv9t4qb1zb2b** | | | cuidmain@HPGICCI1  (TNS V1-V3) | | select CUID\_CUST\_NO ,  CUID\_ID\_... | |
| 3 | | 35 | | 1 | | 3.13 | | 5.28 | | **6z06gcfw39pk d** | | | SQL\*Plus | | SELECT  F.TABLESPACE\_NAME, TO\_... | |

## 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** | **Executio ns** | **Gets per Exec** | **%Tot al** | **CP U**  **Tim e**  **(s)** | **Elapse d Time (s)** | | **SQL Id** | | **SQL Module** | **SQL Text** |
| 3,305,3  63 | 172,329 | 19.18 | 19.85 | 74.57 | 76.41 | | **4vja2k2gdtyu p** | | load\_fnsact@HPGICCI 1 (TNS V1-V3) | insert into ICCICCS  values (:... |
| 2,064,4  14 | 1 | 2,064,414.  00 | 12.40 | 57.31 | 93.50 | | **d8z0u8hgj8xd y** | | cuidmain@HPGICCI1 (TNS V1-V3) | insert into  CUID select CUID\_... |
| 1,826,8  69 | 166,069 | 11.00 | 10.97 | 35.84 | 37.60 | | **7gtztzv329wg 0** | |  | select c.name, u.name from  co... |
| 1,427,6  48 | 172,337 | 8.28 | 8.58 | 12.97 | 13.29 | | **gmn2w09rdx n14** | | load\_oldnewact@HPGI CCI1 (TNS V1-V3) | insert into OLDNEWACT  values ... |
| 1,278,6  67 | 172,329 | 7.42 | 7.68 | 22.85 | 22.94 | | **1dm3bq36vu3 g8** | | load\_fnsact@HPGICCI 1 (TNS V1-V3) | insert into  iccifnsact values... |
| 1,216,3  67 | 1 | 1,216,367.  00 | 7.31 | 42.43 | 50.93 | | **ackxqhnktxn bc** | | cusmmain@HPGICCI1 (TNS V1-V3) | insert into CUSM select  CUSM\_... |
| 1,107,3  05 | 1 | 1,107,305.  00 | 6.65 | 42.01 | 58.04 | | **569r5k05drsj7** | | cumimain@HPGICCI1 (TNS V1-V3) | insert into  CUMI select CUSV\_... |
| 898,868 | 172,983 | 5.20 | 5.40 | 14.28 | 14.34 | | **7wwv1ybs9zg uz** | | load\_fnsact@HPGICCI 1 (TNS V1-V3) | update ICCIFNSACT  set  BORM\_AD... |
| 711,450 | 166,051 | 4.28 | 4.27 | 12.52 | 12.55 | | **chjmy0dxf9m bj** | | icci\_migact@HPGICCI1 (TNS V1-V3) | insert into ICCICCS  values (:... |
| 692,996 | 172,329 | 4.02 | 4.16 | 8.31 | 8.31 | | **38apjgr0p55n** | | load\_fnsact@HPGICCI | update |
|  |  |  |  |  |  | **s** | | 1 (TNS V1-V3) | | ICCICCS set CCSMAXOVE  R... |
| 666,748 | 166,052 | 4.02 | 4.00 | 6.36 | 6.36 | **7v9dyf5r424y h** | | icci\_migact@HPGICCI1 (TNS V1-V3) | | select NEWACTNO  into :b0 from... |
| 345,357 | 172,983 | 2.00 | 2.07 | 7.70 | 7.71 | **5c4qu2zmj3g ux** | | load\_fnsact@HPGICCI 1 (TNS V1-V3) | | select \* from  ICCIPRODCO DE wh... |
| 231,756 | 51,633 | 4.49 | 1.39 | 5.75 | 5.83 | **49ms69srnax zj** | | load\_fnsact@HPGICCI 1 (TNS V1-V3) | | insert into ICCIRPYV  values (... |

## SQL ordered by Reads

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

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;

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Physic al Reads** | | **Executio ns** | | **Reads per Exec** | | **%Tot al** | | **CP U**  **Tim**  **e (s)** | | **Elaps ed Time (s)** | | **SQL Id** | | **SQL Module** | | **SQL Text** |
| 66,286 | | 1 | | 66,286.  00 | | 20.54 | | 57.3  1 | | 93.50 | | **d8z0u8hgj8xd**  **y** | | cuidmain@HPGIC  CI1 (TNS V1-V3) | | insert into CUID select  CUID\_... | |
| 50,646 | | 1 | | 50,646.  00 | | 15.70 | | 3.54 | | 9.70 | | **0yv9t4qb1zb2 b** | | cuidmain@HPGIC CI1 (TNS V1-V3) | | select CUID\_CUST\_NO ,  CUID\_ID\_... | |
| 24,507 | | 1 | | 24,507.  00 | | 7.59 | | 42.0  1 | | 58.04 | | **569r5k05drsj**  **7** | | cumimain@HPGIC  CI1 (TNS V1-V3) | | insert into CUMI select  CUSV\_... | |
| 21,893 | | 1 | | 21,893.  00 | | 6.78 | | 42.4  3 | | 50.93 | | **ackxqhnktxn**  **bc** | | cusmmain@HPGI  CCI1 (TNS V1-V3) | | insert into CUSM select  CUSM\_... | |
| 19,761 | | 1 | | 19,761.  00 | | 6.12 | | 2.14 | | 6.04 | | **a7nh7j8zmfrz w** | | cumimain@HPGIC CI1 (TNS V1-V3) | | select CUSV\_CUST\_NO from  CUMI... | |
| 19,554 | | 1 | | 19,554.  00 | | 6.06 | | 1.27 | | 3.83 | | **38gak8u2qm1**  **1w** | | SQL\*Plus | | select count(\*) from  CUSVAA\_T... | |
| 6,342 | | 1 | | 6,342.0  0 | | 1.97 | | 3.13 | | 35.00 | | **6z06gcfw39p kd** | | SQL\*Plus | | SELECT F.TABLESPACE\_NAM  E, TO\_... | |
| 4,385 | | 1 | | 4,385.0  0 | | 1.36 | | 1.59 | | 2.43 | | **cp5duhcsj72 q0** | | cusmmain@HPGI CCI1 (TNS V1-V3) | | select CUSM\_CUST\_ACCT\_  NO from... | |
| 63 | | 5 | | 12.60 | | 0.02 | | 11.17 | | 14.91 | | **djs2w2f17nw 2z** | |  | | DECLARE job BINARY\_INTEGER := .  .. | |
| 35 | | 1 | | 35.00 | | 0.01 | | 0.08 | | 0.67 | | **1uk5m5qbzj1 vt** | | SQL\*Plus | | BEGIN  dbms\_workload\_reposit ory... | |

## 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;

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Execution s** | **Rows Processe d** | | **Row s per Exec** | | **CPU**  **per Exe c (s)** | | **Elap per Exe**  **c (s)** | | **SQL Id** | | **SQL Module** | | **SQL Text** | |
| 172,983 | 172,329 | | 1.00 | | 0.00 | | 0.00 | | **5c4qu2zmj3gux** | | load\_fnsact@HPGICCI1 (TNS V1-V3) | | select \* from ICCIPRODCODE  wh... | |
| 172,983 | 172,329 | | 1.00 | | 0.00 | | 0.00 | | **7wwv1ybs9zguz** | | load\_fnsact@HPGICCI1 (TNS V1-V3) | | update ICCIFNSACT set  BORM\_AD... | |
| 172,337 | 172,337 | | 1.00 | | 0.00 | | 0.00 | | **gmn2w09rdxn1 4** | | load\_oldnewact@HPGICCI 1 (TNS V1-V3) | | insert into OLDNEWACT  values ... | |
| 172,329 | 172,329 | | 1.00 | | 0.00 | | 0.00 | | **1dm3bq36vu3g 8** | | load\_fnsact@HPGICCI1 (TNS V1-V3) | | insert into iccifnsact  values... | |
| 172,329 | 172,329 | | 1.00 | | 0.00 | | 0.00 | | **38apjgr0p55ns** | | load\_fnsact@HPGICCI1 (TNS V1-V3) | | update ICCICCS set CCSMAXOVER..  . | |
| 172,329 | 6,286 | | 0.04 | | 0.00 | | 0.00 | | **4vja2k2gdtyup** | | load\_fnsact@HPGICCI1 (TNS V1-V3) | | insert into ICCICCS values  (:... | |
| 166,069 | 166,069 | | 1.00 | | 0.00 | | 0.00 | | **7gtztzv329wg0** | |  | | select c.name,  u.name from co... | |
| 166,052 | 166,052 | | 1.00 | | 0.00 | | 0.00 | | **7v9dyf5r424yh** | | icci\_migact@HPGICCI1 (TNS V1-V3) | | select NEWACTNO  into :b0 from... | |
| 166,051 | 166,051 | | 1.00 | | 0.00 | | 0.00 | | **chjmy0dxf9mbj** | | icci\_migact@HPGICCI1  (TNS V1-V3) | | insert into  ICCICCS values | |
|  |  |  | |  | |  | |  | |  | | (:... | |
| 51,740 | 51,740 | 1.00 | | 0.00 | | 0.00 | | **bu8tnqr3xv25q** | | load\_fnsact@HPGICCI1  (TNS V1-V3) | | select count(\*)  into :b0 fro... | |
| 51,633 | 51,633 | 1.00 | | 0.00 | | 0.00 | | **49ms69srnaxzj** | | load\_fnsact@HPGICCI1 (TNS V1-V3) | | insert into ICCIRPYV values  (... | |

## 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** |  | select c.name, u.name from co... | |
| 6,304 | 6,304 | 3.45 | **2ym6hhaq30r73** |  | select type#, blocks, extents,... | |
| 2,437 | 2,438 | 1.33 | **bsa0wjtftg3uw** |  | select file# from file$ where ... | |
| 1,568 | 1,568 | 0.86 | **9qgtwh66xg6nz** |  | update seg$ set type#=:4, bloc... | |
| 1,554 | 1,554 | 0.85 | **aq4js2gkfjru8** |  | update tsq$ set blocks=:3, max... | |
| 444 | 444 | 0.24 | **104pd9mm3fh9p** |  | select blocks, maxblocks, gran... | |
| 421 | 421 | 0.23 | **350f5yrnnmshs** |  | lock table sys.mon\_mods$ in ex... | |
| 421 | 421 | 0.23 | **g00cj285jmgsw** |  | update sys.mon\_mods$ set inser... |
| 86 | 86 | 0.05 | **3m8smr0v7v1m6** |  | INSERT INTO  sys.wri$\_adv\_messa... |
| 81 | 81 | 0.04 | **f80h0xb1qvbsk** |  | SELECT  sys.wri$\_adv\_seq\_msggro... |

## SQL ordered by Sharable Memory

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;

**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

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。

## SQL ordered by Cluster Wait Time

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Clust er Wait Time**  **(s)** | | | **CWT**  **% of Elaps d Time** | | **Elapse d Time(s**  **)** | | **CPU**  **Time( s)** | | **Executio ns** | | | **SQL Id** | | **SQL Module** | | **SQL Text** | |
| 10.96 | | | 11.72 | | 93.50 | | 57.31 | | 1 | | | **d8z0u8hgj8xdy** | | cuidmain@HPGICCI1  (TNS V1-V3) | | insert into CUID select  CUID\_... | |
| 4.21 | | | 7.25 | | 58.04 | | 42.01 | | 1 | | | **569r5k05drsj7** | | cumimain@HPGICCI1  (TNS V1-V3) | | insert into CUMI select  CUSV\_... | |
| 3.62 | | | 7.12 | | 50.93 | | 42.43 | | 1 | | | **ackxqhnktxnb**  **c** | | cusmmain@HPGICCI1  (TNS V1-V3) | | insert into CUSM select  CUSM\_... | |
| 2.39 | | | 6.35 | | 37.60 | | 35.84 | | 166,069 | | | **7gtztzv329wg0** | |  | | select c.name, u.name  from co... | |
| 2.38 | | | 3.12 | | 76.41 | | 74.57 | | 172,329 | | | **4vja2k2gdtyup** | | load\_fnsact@HPGICCI1  (TNS V1-V3) | | insert into ICCICCS  values (:... | |
| 1.64 | | | 16.91 | | 9.70 | | 3.54 | | 1 | | | **0yv9t4qb1zb2b** | | cuidmain@HPGICCI1 (TNS V1-V3) | | select  CUID\_CUST\_NO , CUID\_ID\_... | |
| 1.06 | | | 3.02 | | 35.00 | | 3.13 | | 1 | | | **6z06gcfw39pk d** | | SQL\*Plus | | SELECT  F.TABLESPACE\_NAME, TO\_... | |
| 0.83 | | | 13.76 | | 6.04 | | 2.14 | | 1 | | | **a7nh7j8zmfrzw** | | cumimain@HPGICCI1  (TNS V1-V3) | | select CUSV\_CUST\_NO  from CUMI... | |
| 0.66 | | | 87.90 | | 0.75 | | 0.42 | | 444 | | | **104pd9mm3fh** | |  | | select blocks, | |
|  | |  | | |  | |  | |  | | **9p** | | |  | | maxblocks, gran... | | |
| 0.50 | | 13.01 | | | 3.83 | | 1.27 | | 1 | | **38gak8u2qm11**  **w** | | | SQL\*Plus | | select count(\*) from  CUSVAA\_T... | | |
| 0.50 | | 51.75 | | | 0.96 | | 0.79 | | 1,554 | | **aq4js2gkfjru8** | | |  | | update tsq$ set  blocks=:3, max... | | |
| 0.33 | | 91.11 | | | 0.36 | | 0.33 | | 187 | | **04xtrk7uyhknh** | | |  | | select obj#, type#, ctime,  mti... | | |
| 0.33 | | 2.47 | | | 13.29 | | 12.97 | | 172,337 | | **gmn2w09rdxn**  **14** | | | load\_oldnewact@HPGI  CCI1 (TNS V1-V3) | | insert into OLDNEWACT  values ... | | |
| 0.29 | | 1.26 | | | 22.94 | | 22.85 | | 172,329 | | **1dm3bq36vu3**  **g8** | | | load\_fnsact@HPGICCI1  (TNS V1-V3) | | insert into iccifnsact  values... | | |
| 0.25 | | 10.14 | | | 2.43 | | 1.59 | | 1 | | **cp5duhcsj72q 0** | | | cusmmain@HPGICCI1 (TNS V1-V3) | | select CUSM\_CUST\_ACCT\_N  O from... | | |
| 0.21 | | 27.92 | | | 0.74 | | 0.74 | | 1,568 | | **9qgtwh66xg6n**  **z** | | |  | | update seg$ set  type#=:4, bloc... | | |
| 0.20 | | 3.49 | | | 5.83 | | 5.75 | | 51,633 | | **49ms69srnaxzj** | | | load\_fnsact@HPGICCI1  (TNS V1-V3) | | insert into ICCIRPYV  values (... | | |
| 0.17 | | 1.39 | | | 12.55 | | 12.52 | | 166,051 | | **chjmy0dxf9mb**  **j** | | | icci\_migact@HPGICCI1  (TNS V1-V3) | | insert into ICCICCS  values (:... | | |
| 0.16 | | 57.64 | | | 0.28 | | 0.24 | | 39 | | **cn1gtsav2d5jh** | | | cusvaamain@HPGICCI  1 (TNS V1-V3) | | BEGIN BEGIN IF  (xdb.DBMS... | | |
| 0.14 | | 74.58 | | | 0.19 | | 0.14 | | 121 | | **5ngzsfstg8tmy** | | |  | | select o.owner#,  o.name, o.nam... | | |
| 0.11 | | 64.72 | | | 0.18 | | 0.15 | | 80 | | **78m9ryygp65v 5** | | | cusvaamain@HPGICCI 1 (TNS V1-V3) | | SELECT /\*+  ALL\_ROWS \*/ COUNT(\*... | | |
| 0.11 | | 94.54 | | | 0.12 | | 0.01 | | 17 | | **bwt0pmxhv7q**  **k7** | | |  | | delete from con$ where  owner#=... | | |
| 0.11 | | 80.26 | | | 0.14 | | 0.14 | | 327 | | **53saa2zkr6wc**  **3** | | |  | | select intcol#, nvl(pos#,  0), ... | | |
| 0.08 | | 19.20 | | | 0.42 | | 0.24 | | 1 | | **d92h3rjp0y217** | | |  | | begin prvt\_hdm.auto\_execute(  :... | | |
| 0.07 | | 54.97 | | | 0.13 | | 0.13 | | 83 | | **7ng34ruy5awx**  **q** | | |  | | select i.obj#, i.ts#,  i.file#,... | | |
| 0.06 | | 5.22 | | | 1.13 | | 0.72 | | 77 | | **0hhmdwwgxb**  **w0r** | | |  | | select obj#, type#,  flags, ... | | |
| 0.06 | | 86.50 | | | 0.06 | | 0.06 | | 45 | | **a2any035u1qz** | | |  | | select owner#, name | | |
|  | |  | | |  | |  | |  | | **1** | | |  | | from con$... | | |
| 0.06 | | 8.19 | | | 0.67 | | 0.08 | | 1 | | **1uk5m5qbzj1vt** | | | SQL\*Plus | | BEGIN  dbms\_workload\_reposit ory... | | |
| 0.04 | | 75.69 | | | 0.06 | | 0.06 | | 87 | | **6769wyy3yf66f** | | |  | | select pos#, intcol#,  col#, sp... | | |
| 0.04 | | 48.05 | | | 0.09 | | 0.07 | | 7 | | **0pvtkmrrq8us**  **g** | | |  | | select file#, block# from  seg... | | |
| 0.04 | | 8.84 | | | 0.40 | | 0.40 | | 6,304 | | **2ym6hhaq30r7**  **3** | | |  | | select type#, blocks,  extents,... | | |
| 0.03 | | 28.15 | | | 0.12 | | 0.12 | | 49 | | **b52m6vduutr8j** | | |  | | delete from  RecycleBin$ ... | | |
| 0.03 | | 66.23 | | | 0.05 | | 0.05 | | 85 | | **1gu8t96d0bdm**  **u** | | |  | | select t.ts#, t.file#,  t.block... | | |
| 0.03 | | 67.03 | | | 0.05 | | 0.05 | | 38 | | **btzq46kta67dz** | | | DBMS\_SCHEDULER | | update obj$ set obj#=:6,  type#... | | |
| 0.02 | | 66.73 | | | 0.04 | | 0.04 | | 86 | | **3m8smr0v7v1**  **m6** | | |  | | INSERT INTO  sys.wri$\_adv\_messa... | | |
| 0.02 | | 26.94 | | | 0.09 | | 0.09 | | 38 | | **0k8h617b8guh**  **f** | | |  | | delete from  RecycleBin$ ... | | |
| 0.02 | | 76.76 | | | 0.03 | | 0.03 | | 51 | | **9vtm7gy4fr2ny** | | |  | | select con# from  con$ where ow... | | |
| 0.02 | | 51.91 | | | 0.05 | | 0.05 | | 84 | | **83taa7kaw59c**  **1** | | |  | | select name, intcol#,  segcol#,... | | |
| 0.02 | | 0.15 | | | 14.91 | | 11.17 | | 5 | | **djs2w2f17nw2**  **z** | | |  | | DECLARE job  BINARY\_INTEGER := ... | | |
| 0.02 | | 2.12 | | | 1.00 | | 0.99 | | 8,784 | | **501v412s13r4**  **m** | | | load\_fnsact@HPGICCI1  (TNS V1-V3) | | update ICCIFNSACT set  BORM\_FA... | | |
| 0.02 | | 53.82 | | | 0.03 | | 0.03 | | 39 | | **bdv0rkkssq2j**  **m** | | | cusvaamain@HPGICCI  1 (TNS V1-V3) | | SELECT count(\*) FROM  user\_poli... | | |
| 0.01 | | 0.10 | | | 14.34 | | 14.28 | | 172,983 | | **7wwv1ybs9zg**  **uz** | | | load\_fnsact@HPGICCI1  (TNS V1-V3) | | update ICCIFNSACT set  BORM\_AD... | | |
| 0.01 | | 8.29 | | | 0.16 | | 0.13 | | 421 | | **g00cj285jmgs w** | | |  | | update  sys.mon\_mods$ set inser... | | |
| 0.01 | | 1.65 | | | 0.56 | | 0.54 | | 2 | | **84qubbrsr0kfn** | | |  | | insert into wrh$\_latch  (snap... | | |
| 0.01 | | 22.33 | | | 0.04 | | 0.02 | | 26 | | **44au3v5mzpc1**  **c** | | | load\_curmmast@HPGI  CCI1 (TNS V1-V3) | | insert into  ICCICURMMAST valu... | | |
| 0.01 | | | | 0.08 | | 7.71 | | 7.70 | | | 172,983 | | **5c4qu2zmj3gu**  **x** | | load\_fnsact@HPGICCI1  (TNS V1-V3) | | select \* from  ICCIPRODCODE wh... | |
|  | | | | | | | | | | |

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** | | |
| 04xtrk7  uyhknh | | 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 | | |
| 0hhmd  wwgxb w0r | | select obj#, type#, flags, related, bo, purgeobj, con# from RecycleBin$ where ts#=:1 and to\_number(bitand(flags, 16)) = 16 order by dropscn | | |
| 0k8h61  7b8guh f | | delete from RecycleBin$ where purgeobj=:1 | | |
| 0pvtkm rrq8us  g | | select file#, block# from seg$ where type# = 3 and ts# = :1 | | | |
| 0yv9t4 qb1zb2  b | | select CUID\_CUST\_NO , CUID\_ID\_TYPE , CUID\_ID\_RECNO from CUID\_TMP where CHGFLAG='D' | | | |
| 104pd9  mm3fh 9p | | select blocks, maxblocks, grantor#, priv1, priv2, priv3 from tsq$ where ts#=:1 and user#=:2 | | | |
| 1crajpb 7j5tyz | | 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 | | | |
| 1dm3b  q36vu3 g8 | | 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) | | | |
| 1gu8t9 6d0bd mu | | 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# (+) | | | |
| 1uk5m 5qbzj1  vt | | BEGIN dbms\_workload\_repository.create\_snapshot; END; | | | |
| 2ym6h haq30r  73 | | 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 | | | |
| 350f5yr nnmsh  s | | lock table sys.mon\_mods$ in exclusive mode nowait | | | |
| 38apjgr  0p55ns | | update ICCICCS set CCSMAXOVERDUE=GREATEST(:b0, CCSMAXOVERDUE) where FNSACTNO=:b1 | | | |
| 38gak8 u2qm1  1w | | select count(\*) from CUSVAA\_TMP | | | |
| 3m8sm r0v7v1  m6 | | INSERT INTO sys.wri$\_adv\_message\_groups (task\_id, id, seq, message#, fac, hdr, lm, nl, p1, p2, p3, p4, p5) VALUES (:1, :2, :3, :4, :5, :6, :7, :8, :9, :10, :11, :12, :13) | | | |
| 44au3v 5mzpc  1c | | insert into ICCICURMMAST values (:b0, :b1, :b2) | | | |
| 49ms6 9srnax zj | | insert into ICCIRPYV values (:b0, :b1, :b2, :b3, :b4, :b5, :b6, :b7, :b8, :b9, :b10, :b11, :b12, :b13, :b14, :b15, :b16, :b17, :b18, :b19, :b20, :b  21, :b22, :b23, :b24, :b25, :b26, :b27, :b28, :b29, :b30, :b31, :b32, :b33, :b34, :b35, :b36, :b37, :b38, :b39, :b4  0, :b41, :b42, :b43, :b44, :b45, :b46, :b47, :b48, :b49, :b50, :b51) | | | |
| 4vja2k 2gdtyu  p | | insert into ICCICCS values (:b0, '////////////////////////', 0, 0, 0, 0, 0, ' ', 0, 0, 0, ' ', '0', null ) | | | |
| 501v41  2s13r4 m | | update ICCIFNSACT set BORM\_FACILITY\_NO=:b0 where BORM\_MEMB\_CUST\_AC=:b1 | | | |
| 53saa2 zkr6wc  3 | | select intcol#, nvl(pos#, 0), col#, nvl(spare1, 0) from ccol$ where con#=:1 | | | |
| 569r5k  05drsj7 | | insert into CUMI select CUSV\_CUST\_NO , CUSV\_EDUCATION\_CODE , CHGDATE from CUMI\_TMP where  CHGFLAG<>'D' | | | |
| 5c4qu2  zmj3gu x | | select \* from ICCIPRODCODE where PRODCODE=to\_char(:b0) | | | |
| 5ngzsf stg8tm  y | | select o.owner#, o.name, o.namespace, o.remoteowner, o.linkname, o.subname, o.dataobj#, o.flags from obj$ o where o.obj#=:1 | | | |
| 6769w  yy3yf6 6f | | select pos#, intcol#, col#, spare1, bo#, spare2 from icol$ where obj#=:1 | | | |
| 6z06gc fw39pk d | | 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$PARAMETER WHERE NAME = 'db\_block\_size')/1024) ) FREE\_SPACE FROM DBA\_FREE\_SPACE GROUP BY TABLESPACE\_NAME ) F, ( SELECT TABLESPACE\_NAME, ROUND (SUM (BYTES/1048576)) TOTAL\_SPACE FROM DBA\_DATA\_FILES GROUP BY TABLESPACE\_NAME ) T WHERE  F.TABLESPACE\_NAME = T.TABLESPACE\_NAME | | | |
| 78m9ry ygp65v  5 | | SELECT /\*+ ALL\_ROWS \*/ COUNT(\*) FROM ALL\_POLICIES V WHERE V.OBJECT\_OWNER = :B3 AND V.OBJECT\_NAME = :B2 AND (POLICY\_NAME LIKE '%xdbrls%' OR POLICY\_NAME LIKE '%$xd\_%') AND  V.FUNCTION = :B1 | | | |
| 7gtztzv  329wg 0 | | 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# | | | |
| 7ng34r uy5aw  xq | | 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# | | | |
| 7v9dyf 5r424y  h | | select NEWACTNO into :b0 from OLDNEWACT where OLDACTNO=:b1 | | | |
| 7wwv1 ybs9zg uz | | 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 | | | |
| 83taa7 kaw59 c1 | | 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# | | | |
| 84qubb rsr0kfn | | 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 | | | |
| 9qgtwh 66xg6n  z | | 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 | | | |
| 9vtm7g  y4fr2ny | | select con# from con$ where owner#=:1 and name=:2 | | | |
| a2any0 35u1qz  1 | | select owner#, name from con$ where con#=:1 | | | |
| a7nh7j  8zmfrz w | | select CUSV\_CUST\_NO from CUMI\_TMP where CHGFLAG='D' | | | |
| ackxqh nktxnb  c | | insert into CUSM select CUSM\_CUST\_ACCT\_NO , CUSM\_STAT\_POST\_ADD\_NO , CHGDATE from CUSM\_TMP where CHGFLAG<>'D' | | | |
| aq4js2  gkfjru8 | | update tsq$ set blocks=:3, maxblocks=:4, grantor#=:5, priv1=:6, priv2=:7, priv3=:8 where ts#=:1 and user#=:2 | | | |
| b52m6  vduutr8 j | | delete from RecycleBin$ where bo=:1 | | | |
| bdv0rk  kssq2j | | SELECT count(\*) FROM user\_policies o WHERE o.object\_name = :tablename AND (policy\_name LIKE  '%xdbrls%' OR policy\_name LIKE '%$xd\_%') AND o.function='CHECKPRIVRLS\_SELECTPF' | | | |
| m | | | |  |
| bsa0wj  tftg3uw | | | | select file# from file$ where ts#=:1 |
| btzq46 kta67d z | | | | update obj$ set obj#=:6, type#=:7, ctime=:8, mtime=:9, stime=:10, status=:11, dataobj#=:13, flags=:14, oid$=:15, spare1=:16, spare2=:17 where owner#=:1 and name=:2 and namespace=:3 and(remoteowner=:4 or remoteowner is null and :4 is null)and(linkname=:5 or linkname is null and :5 is null)and(subname=:12 or  subname is null and :12 is null) |
| bu8tnq r3xv25  q | | | | select count(\*) into :b0 from ICCIFNSACT where BORM\_MEMB\_CUST\_AC=:b1 |
| bwt0p  mxhv7 qk7 | | | | delete from con$ where owner#=:1 and name=:2 |
| chjmy0 dxf9mb  j | | | | insert into ICCICCS values (:b0, :b1, :b2, :b3, :b4, :b5, :b6, :b7, :b8, :b9, :b10, :b11, :b12, :b13) |
| cn1gts av2d5j h | | | | BEGIN BEGIN IF (xdb.DBMS\_XDBZ0.is\_hierarchy\_enabled\_internal(sys.dictionary\_obj\_owner, sys.dictionary\_obj\_name, sys.dictionary\_obj\_owner)) THEN xdb.XDB\_PITRIG\_PKG.pitrig\_truncate(sys.dictionary\_obj\_owner, sys.dictionary\_obj\_name); END IF; EXCEPTION WHEN OTHERS THEN null; END; BEGIN IF  (xdb.DBMS\_XDBZ0.is\_hierarchy\_enabled\_internal(sys.dictionary\_obj\_owner, sys.dictionary\_obj\_name, sys.dictionary\_obj\_owner, xdb.DBMS\_XDBZ.IS\_ENABLED\_RESMETADATA)) THEN xdb.XDB\_PITRIG\_PKG.pitrig\_dropmetadata(sys.dictionary\_obj\_owner, sys.dictionary\_obj\_name); END IF;  EXCEPTION WHEN OTHERS THEN null; END; END; |
| cp5duh csj72q  0 | | | | select CUSM\_CUST\_ACCT\_NO from CUSM\_TMP where CHGFLAG='D' |
| d8z0u8 hgj8xd  y | | | | insert into CUID select CUID\_CUST\_NO , CUID\_ID\_MAIN , CUID\_ID\_TYPE , CUID\_ID\_RECNO , CUID\_ID\_NUMBER , CHGDATE from CUID\_TMP where CHGFLAG<>'D' |
| d92h3rj  p0y217 | | | | begin prvt\_hdm.auto\_execute( :db\_id, :inst\_id, :end\_snap ); end; |
| djs2w2 f17nw2  z | | | | DECLARE job BINARY\_INTEGER := :job; next\_date DATE := :mydate; broken BOOLEAN := FALSE; BEGIN statspack.snap; :mydate := next\_date; IF broken THEN :b := 1; ELSE :b := 0; END IF; END; |
| f80h0x b1qvbs  k | | | | SELECT sys.wri$\_adv\_seq\_msggroup.nextval FROM dual |
| g00cj2 85jmgs  w | | | | update sys.mon\_mods$ set inserts = inserts + :ins, updates = updates + :upd, deletes = deletes + :del, flags = (decode(bitand(flags, :flag), :flag, flags, flags + :flag)), drop\_segments = drop\_segments + :dropseg,  timestamp = :time where obj# = :objn |
| gmn2w 09rdxn  14 | | insert into OLDNEWACT values (:b0, :b1) | | |

# Instance Activity Statistics

* + **Instance Activity Stats**
  + **Instance Activity Stats - Absolute Values**
  + **Instance Activity Stats - Thread Activity**

## 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 |
| 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 |

## 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 |

## 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 |

# IO Stats

* + **Tablespace IO Stats**
  + **File IO Stats**

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。

1. 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。

## File IO Stats

* + **ordered by Tablespace, File**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Tablespace** | | **Filename** | | **Read s** | | **Av Reads/**  **s** | | **Av Rd(ms**  **)** | | **Av Blks/R**  **d** | | **Writes** | | **Av Writes/s** | | **Buffe r**  **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\_UNDO010  1 | | 10 | | 0 | | 12.00 | | 1.00 | | 57,771 | | 12 | | 625 | | 0.02 | | |
| UNDOTBS2 | | /dev/rora\_UNDO020  1 | | 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 | | |

# 十一、Advisory Statistics

* + **Instance Recovery Stats**
  + **Buffer Pool Advisory**
  + **PGA Aggr Summary**
  + **PGA Aggr Target Stats**
  + **PGA Aggr Target Histogram**
  + **PGA Memory Advisory**
  + **Shared Pool Advisory**
  + **SGA Target Advisory**
  + **Streams Pool Advisory**
  + **Java Pool Advisory**

## 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。

## 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 |

## 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 |

## 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 |

# 十三、Latch Statistics

* + **Latch Activity**
  + **Latch Sleep Breakdown**
  + **Latch Miss Sources**
  + **Parent Latch Statistics**
  + **Child Latch Statistics**

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：

增大 shared pool

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 | |  | |

# 十五、Dictionary Cache Statistics

* + **Dictionary Cache Stats**
  + **Dictionary Cache Stats (RAC)**

/\* 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 | |

# 十六、Memory Statistics

* + **Process Memory Summary**
  + **SGA Memory Summary**
  + **SGA breakdown difference**

## 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 | |  | |

# 十八、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 |

## 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 | |  | |

# 