9. 等待事件

很长时间以来，通过什么样的手段来衡量数据库的状况，发现数据库的问题，优化数据库的性能一直是人们广为争论的话题。从 Oracle 7.0.12 开始，Oracle 引入了等待事件，随即等待事件成为了数据库性能优化的一个重要指导。

当一个进程连接到数据库之后，进程所经历的种种等待就开始被记录，并且通过一系列的动态性能视图进行展现。通过等待事件用户可以很快地发现数据库的性能瓶颈，从而进行针对性优化和分析。本章将着重介绍等待事件在 Oracle 研究及优化过程中的作用。

9.1 等待事件的源起

等待事件的概念是在 Oracle 7.0.12 中引入的，大致有 100 个等待事件。在 Oracle 9.0 中这个数目增加到了大约 150 个，在 Oracle 8i 中大约有 220 个事件，在 Oracle 9iR2 中大约有 400 个等待事件，在 Oracle 10gR2 中大约有 874 个等待事件，而在最近的 Oracle 11gR1 中，等待事件的数目已经接近了 1000 个。

虽然不同的版本和组件安装可能会有不同数目的等待事件，但是这些等待事件都可以通过查询 V$EVENT_NAME 视图获得：

```
SQL> select * from v$version where rownum <2;
BANNER
--------------------------------------------------------------------------------
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
SQL> select count(*) from v$event_name;
  COUNT(*)
----------
       961
```

研究 Oracle 的等待事件，V$EVENT_NAME 视图是一个很好的开始，这个视图记录着当前数据库支持的等待事件及其基本信息。

Oracle 的等待事件，主要可以分为两类，即空闲（idle）等待事件和非空闲（non-idle）等待事件。空闲事件指 Oracle 正等待某种工作，在诊断和优化数据库的时候，我们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的时候应该关注与研究的。

在 Oracle 10g 之前，Oracle 的 Statspack 会创建一个视图 stats$idle_event 记录空闲等待事件：

```
SQL> select * from stats$idle_event;
EVENT
```

```
----------------------------------------------------------------
smon timer
pmon timer
rdbms ipc message
Null event
parallel query dequeue
pipe get
client message
SQL*Net message to client
SQL*Net message from client
SQL*Net more data from client
dispatcher timer
virtual circuit status
lock manager wait for remote message
PX Idle Wait
wakeup time manager


15 rows selected.
```

从 Oracle 10g 开始，Oracle 对等待事件进行了更为详细的分类，V$EVENT_NAME 视图也增加了相关分类的字段：

```
SQL> desc v$event_name
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EVENT#                                             NUMBER
 EVENT_ID                                           NUMBER
 NAME                                               VARCHAR2(64)
 PARAMETER1                                         VARCHAR2(64)
 PARAMETER2                                         VARCHAR2(64)
 PARAMETER3                                         VARCHAR2(64)
 WAIT_CLASS_ID                                      NUMBER
 WAIT_CLASS#                                        NUMBER
 WAIT_CLASS                                         VARCHAR2(64)
```

V$EVENT_NAME 视图中的 PARAMETER1、PARAMETER2、PARAMETER3 非常重要，对于不同的等待事件参数其意义各不相同：

```
SQL> select name,PARAMETER1,PARAMETER2,PARAMETER3 from v$event_name
  2  where name ='db file scattered read';
NAME                     PARAMETER1      PARAMETER2      PARAMETER3
----------------------- --------------- --------------- ---------------
db file scattered read                 file#           block#          blocks
```

看一下 Oracle 11gR1 中主要分类及各类等待事件的个数：

```
SQL> SELECT   wait_class#, wait_class_id, wait_class, COUNT (*) AS "count"
  2  FROM v$event_name GROUP BY wait_class#, wait_class_id, wait_class
  3  ORDER BY wait_class#;
WAIT_CLASS# WAIT_CLASS_ID WAIT_CLASS                  count
----------- ------------- -------------------- ----------
          0    1893977003 Other                       632
          1    4217450380 Application                  15
          2    3290255840 Configuration                21
          3    4166625743 Administrative               51
          4    3875070507 Concurrency                  26
          5    3386400367 Commit                        2
          6    2723168908 Idle                         80
          7    2000153315 Network                      35
          8    1740759767 User I/O                     22
          9    4108307767 System I/O                   23
         10    2396326234 Scheduler                     3
         11    3871361733 Cluster                      47
         12     644977587 Queueing                      4
13 rows selected.
```

也可以通过查询 V$SYSTEM_WAIT_CLASS 视图获得各类主要等待事件的等待时间和等待次数等信息，通过分类以及统计信息，可以很直观地快速获得数据库的整体印象，在以下输出中，可以看出数据库的主要等待消耗在 User I/O 操作上：

```
SQL> select * from v$system_wait_class order by time_waited;
WAIT_CLASS_ID WAIT_CLASS# WAIT_CLASS     TOTAL_WAITS  TIME_WAITED
------------- ----------- -------------------- ----------- -----------
   3875070507           4 Concurrency                 8433         751
   4217450380           1 Application                  366        2558
   1893977003           0 Other                      15690       14765
   3386400367           5 Commit                     30520       49246
   3290255840           2 Configuration               5701       102057
   2000153315           7 Network                  6261634      103300
   4108307767           9 System I/O               1258815      1613868
   1740759767           8 User I/O                 9027852      3358285
   2723168908           6 Idle                     4402568    794064698
9 rows selected
```

从 Oracle 10g 开始，可以通过如下查询来首先了解数据库的空闲等待事件：

```
select name,wait_class from v$event_name where wait_class='Idle';
```

在 Oracle 11g 中，空闲等待已经增加到 80 个左右。

9.2 从等待发现瓶颈

等待事件所以为众多DBA所关注与研究，是因为通过等待事件可以迅速发现数据库瓶颈，并及时解决问题。在网上，我曾经发起过一个讨论，让大家“列举你认为最重要的9个动态性能视图”，很多人的回复里都选择了和等待事件相关的几个视图，它们是 V$SESSION、V$SESSION_WAIT 和 V$SYSTEM_EVENT。

来看一下这几个视图的作用及重要意义。

- V$SESSION 视图记录的是数据库当前连接的 Session 信息。
- V$SESSION_WAIT 视图记录的是当前数据库连接的活动 Session 正在等待的资源或事件信息。
- 由于 V$SESSION 记录的是动态信息，和 Session 的生命周期相关，并不记录历史信息，所以 Oracle 提供另外一个视图 V$SYSTEM_EVENT 来记录数据库自启动以来所有等待事件的汇总信息。通过 V$SYSTEM_EVENT 视图，可以迅速地获得数据库运行的总体概况。

9.2.1 V$SESSION 和 V$SESSION_WAIT

由于 V$SESSION 记录当前连接数据库的 Session 信息，而 V$SESSION_WAIT 视图记录这些 Session 的等待，很多时候我们要联合这两个视图进行查询以获取更多的诊断信息。从 Oracle 10g 开始，为了方便用户，Oracle 开始将这两个视图进行整合。

在 Oracle 10gR1 中，Oracle 在 V$SESSION 中增加关于等待事件的字段，实际上也就是把原来 V$SESSION_WAIT 视图中的所有字段全部整合到了 V$SESSION 视图中（如果进一步研究你会发现，实际上 V$SESSION 的底层查询语句及 X$表已经有了变化）。这一变化使得查询得以简化，但是也使得 V$SESSION_WAIT 开始变得多余。

此外 V$SESSION 中还增加了 BLOCKING_SESSION 等字段，以前需要通过 dba_waiters 等视图才能获得的信息，现在也可以直接从 V$SESSION 中得到了。

在 Oracle 10gR2 中，Oracle 又为 V$SESSION 增加了额外几个字段：SERVICE_NAME、SQL_TRACE、SQL_TRACE_WAITS、SQL_TRACE_BINDS。这几个字段显示当前 Session 连接方式及是否启用了 SQL_TRACE 跟踪等。

在 Oracle 11gR1 中，V$SESSION 的内容进一步增强，增加了很多新的字段，比如 SQL_EXEC_START、SQL_EXEC_ID 用于记录 SQL 执行的开始时间及执行 ID（相应的还有 PREV_EXEC_START、PREV_EXEC_ID 等字段）。

```
SQL> alter session set nls_date_format='yyyy-mm-dd hh24:mi:ss';
Session altered.
SQL> select sid,username,sql_exec_start,sql_exec_id from v$session
  2 where sql_exec_id is not null;
       SID USERNAME                       SQL_EXEC_START      SQL_EXEC_ID
---------- ------------------------------ ------------------- -----------
       126 EYGLE                          2008-07-16 10:44:25    16777236
       130 SYS                            2008-07-16 10:44:25    16777217
```

在新的数据库版本中，Oracle 在小处动的手脚也是非常多的，而无疑这些小手脚会给用户的管理维护带来极大的方便。以下是 Oracle 9iR2 中 V$SESSION_WAIT 视图的结构：

```
SQL> desc v$session_wait
 Name              Null?    Type
 -------- -------- ----------------------------
 SID                         NUMBER
 SEQ#                        NUMBER
 EVENT                      VARCHAR2(64)
 P1TEXT                     VARCHAR2(64)
 P1                           NUMBER
 P1RAW                      RAW(4)
 P2TEXT                     VARCHAR2(64)
 P2                          NUMBER
 P2RAW                      RAW(4)
 P3TEXT                     VARCHAR2(64)
 P3                          NUMBER
 P3RAW                      RAW(4)
 WAIT_TIME                  NUMBER
 SECONDS_IN_WAIT           NUMBER
 STATE                       VARCHAR2(19)
```

其中，event 代表等待事件的名称，p<n>text 用以描述具体的参数，p<n>分别代表以十进制定义的参数（parameter）参数值，p<n>Raw 是以十六进制表示的参数值。对于不同 event，具体参数表示的含义也不相同，可以通过 v$event_name 视图来查看这些参数的定义。

9.2.2　V$SESSION_EVENT 和 V$SYSTEM_EVENT

上一节提到的 V$SESSION 及 V$SESSION_WAIT 视图记录了活动会话当前正在发生的等待，但是要知道一个活动会话在其生命周期只能可能经历很多等待，这些等待通过 V$SESSION_EVENT 视图记录。但是需要注意的是，这个视图记录的是累积信息，同一会话对于同一事件发生的多次等待会被累计。以下是一个会话的等待事件输出：

```
SQL> select sid,event,time_waited,time_waited_micro
  2 from v$session_event where sid=546 order by 3;
       SID EVENT                                    TIME_WAITED TIME_WAITED_MICRO
---------- ---------------------------------------- ----------- -----------------
       546 log file sync                                      0              3084
       546 latch: library cache lock                          1              7171
       546 latch: library cache pin                           2             16112
       546 latch: library cache                               4             43621
       546 buffer busy waits                                  9             86652
```

```
        546 latch: shared pool                                10          103100
        546 db file scattered read                            12          123146
        546 latch: cache buffers chains                       17          165332
        546 log file switch completion                        57          572292
        546 events in waitclass Other                         88          877450
        546 db file sequential read                         1471        14713213
        546 os thread startup                              14224       142236350
        546 control file sequential read                   49067       490672360
        546 rdbms ipc message                         1021388726      1.0214E+13
已选择 14 行。
```

V$SESSION_EVENT 的信息和会话生命周期相关，这些信息同时会被累积到 V$SYSTEM_EVENT 视图作为数据库整体等待数据保存，比如：

```
SQL> select event,total_waits,time_waited,average_wait from v$system_event
  2  where event='latch: shared pool';
EVENT                          TOTAL_WAITS TIME_WAITED AVERAGE_WAIT
------------------------------ ----------- ----------- ------------
latch: shared pool                   13931        2770           .2
```

但是注意，V$SESSION_EVENT 或者 V$SYSTEM_EVENT 视图的累积信息以及关于等待的平均计算，使我们无法得知个别等待消耗的时间长短。

为了解决这一问题，Oracle 10g 引入了一个新的视图 v$event_histogram，通过这个视图可以看到等待事件的柱状图分布，从而可以对一个等待事件的具体分布有进一步的了解，在以下查询输出中可以看到，Shared Pool Latch 的竞争主要是 10 毫秒以内的短时竞争，但是注意等待时间在 256 毫秒左右的等待也有 5 次，长时间的 Latch 竞争是在数据库优化时需要认真关注的：

```
SQL> SELECT event, wait_time_milli, wait_count
  2  FROM   v$event_histogram WHERE  event = 'latch: shared pool';
EVENT                          WAIT_TIME_MILLI WAIT_COUNT
------------------------------ --------------- ----------
latch: shared pool                           1       8604
latch: shared pool                           2       2248
latch: shared pool                           4       1208
latch: shared pool                           8        781
latch: shared pool                          16        400
latch: shared pool                          32        150
latch: shared pool                          64         49
latch: shared pool                         128         21
latch: shared pool                         256          5
已选择 9 行。
```

9.2.3　Oracle 11g 实时 SQL 监控

前面提到，在 Oracle Database 11g 中，v$session 视图增加了一些新的字段，这其中包括 SQL_EXEC_START 和 SQL_EXEC_ID，这两个字段实际上代表了 Oracle 11g 的一个新特性：实时的 SQL 监视（Real Time SQL Monitoring）。

在 Oracle 11g 之前的版本，长时间运行的 SQL 可以通过监控 v$session_longops 来观察，当某个操作执行时间超过 6 秒，就会被记录在 v$session_longops 中，通常可以监控到全表扫描、全索引扫描、哈希联接、并行查询等操作；而在 Oracle 11g 中，当 SQL 并行执行时，会立即被实时监控到，或者当 SQL 单进程运行时，如果消耗超过 5 秒的 CPU 或 I/O 时间，它也会被监控到。监控数据被记录在 V$SQL_MONITOR 视图中，当然也可以通过 Oracle 11g 新增的 package DBMS_MONITOR 来主动对 SQL 执行监控部署。

来看一下主要视图 V$SQL_MONITOR 的结构：

```
SQL> desc v$sql_monitor
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 KEY                                                NUMBER
 STATUS                                             VARCHAR2(19)
 FIRST_REFRESH_TIME                                 DATE
 LAST_REFRESH_TIME                                  DATE
 REFRESH_COUNT                                      NUMBER
 SID                                                NUMBER
 PROCESS_NAME                                       VARCHAR2(5)
 SQL_ID                                             VARCHAR2(13)
 SQL_EXEC_START                                     DATE
 SQL_EXEC_ID                                        NUMBER
 SQL_PLAN_HASH_VALUE                                NUMBER
 SQL_CHILD_ADDRESS                                  RAW(4)
 SESSION_SERIAL#                                    NUMBER
 PX_SERVER#                                         NUMBER
 PX_SERVER_GROUP                                    NUMBER
 PX_SERVER_SET                                      NUMBER
 PX_QCINST_ID                                       NUMBER
 PX_QCSID                                           NUMBER
 ELAPSED_TIME                                       NUMBER
 CPU_TIME                                           NUMBER
 FETCHES                                            NUMBER
 BUFFER_GETS                                        NUMBER
 DISK_READS                                         NUMBER
 DIRECT_WRITES                                      NUMBER
```

```
APPLICATION_WAIT_TIME                                     NUMBER
CONCURRENCY_WAIT_TIME                                     NUMBER
CLUSTER_WAIT_TIME                                         NUMBER
USER_IO_WAIT_TIME                                         NUMBER
PLSQL_EXEC_TIME                                           NUMBER
JAVA_EXEC_TIME                                            NUMBER
```

注意这里的 SQL_EXEC_ID 就是 V$SESSION 视图中新增字段的来源。这个视图还记录了 SQL 的 CPU_TIME 以及 BUFFER_GETS 等重要信息，对于诊断 SQL 性能问题具有极大的帮助。结合 V$SQL_MONITOR 视图与 V$SQL_PLAN_MONITOR 视图可以进一步查询 SQL 的执行计划等信息。联合一些其他视图，如 v$active_session_history、v$session、v$session_longops、v$sql、v$sql_plan 等，可以获得关于 SQL 的更多信息。

V$SQL_MONITOR 收集的信息每秒刷新一次，接近实时，当 SQL 执行完毕，信息并不会立即从 v$sql_monitor 中删除，至少会保留 1 分钟，v$sql_plan_monitor 视图中的执行计划信息也是每秒更新一次，当 SQL 执行完结，它们同样至少被保留 1 分钟。

实时 SQL 监控需要 statistics_level 初始化参数设置为 TYPICAL 或 ALL：

```
SQL> show parameter statistics_level
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
statistics_level                     string      TYPICAL
SQL> SELECT statistics_name,session_status,system_status,activation_level,session_settable
  2  FROM v$statistics_level WHERE statistics_name = 'SQL Monitoring';
STATISTICS_NAME                SESSION_ SYSTEM_S ACTIVAT SES
------------------------------ -------- -------- ------- ---
SQL Monitoring                 ENABLED  ENABLED  TYPICAL YES
```

同时 CONTROL_MANAGEMENT_PACK_ACCESS 参数必须是 DIAGNOSTIC+TUNING（这是缺省设置）：

```
SQL> show parameter control_manage
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
control_management_pack_access       string      DIAGNOSTIC+TUNING
```

在如上设置下，数据库会启动自动的实时 SQL 监控，Oracle 还提供 Hints 可以强制制定对 SQL 执行监控或者不允许执行监控，这两个 Hints 是 monitor 与 no_monitor。

强制对某个 SQL 使用实时监控可以如下改写 SQL：

select /+ monitor */ count(*) from emp where sal > 5000;*

指定不执行实时监控：

select /+ no_monitor */ count(*) from emp where sal >5000;*

查看数据库中已经生成的监控信息可以使用 DBMS_SQLTUNE 包来实现：

```
set long 10000000
set longchunksize 10000000
```

```
set linesize 200
select dbms_sqltune.report_sql_monitor from dual;
```

以下是一个 Oracle Database 11g 生产环境中的查询输出，系统中目前记录了一条 SQL 的监视信息。这条 SQL 使用了全表扫描，看起来缺乏索引，Oracle 现在自动为用户记录了详细的信息：

```
SQL> set long 10000000
SQL> set longchunksize 10000000
SQL> set linesize 200
select dbms_sqltune.report_sql_monitor from dual;
SQL> REPORT_SQL_MONITOR
-------------------------------------------------------------------------------------
SQL Monitoring Report

SQL Text
----------------------------------------------------------------------------------------
select * from forecast where cityid = '886' and to_char(forecastdate,'YYYY/MM/DD') =
'2008/07/15'
----------------------------------------------------------------------------------------


Global Information
 Status              :  DONE (ALL ROWS)
 Instance ID         :  1
 Session ID          :  71

REPORT_SQL_MONITOR
----------------------------------------------------------------------------------------
 SQL ID              :  1rrshaasrsa1z
 SQL Execution ID    :  16777218
 Plan Hash Value     :  2831319728
 Execution Started   :  07/15/2008 15:47:31
 First Refresh Time  :  07/15/2008 15:47:35
 Last Refresh Time   :  07/15/2008 15:47:37
-------------------------------------------------------------------
| Elapsed |   Cpu   |    IO    |  Other   | Fetch | Buffer | Reads |
| Time(s) | Time(s) | Waits(s) | Waits(s) | Calls |  Gets  |       |
--------------------------------------------------------------------
|    3.51 |    0.67 |     0.00 |     2.84 |     1 |   8350 |  8203 |
-------------------------------------------------------------------
SQL Plan Monitoring Details
```

```
===================================================================================
| Id |      Operation       |   Name    |  Rows   | Cost |    Time     | Start  | Starts |
|    |                      |           | (Estim) |      | Active(s) | Active |        |
===================================================================================
|  0 | SELECT STATEMENT     |           |         | 2478 |           1 |     +6 |      1 |
|  1 |   TABLE ACCESS FULL  | FORECAST |      25 | 2478 |           5 |     +2 |      1 |
-----------------------------------------------------------------------------------
```

对于数据库中已经捕获的 SQL，通过其 SQL_ID，使用 DBMS_SQLTUNE 程序包中的 REPORT_SQL_MONITOR 函数，我们可以生成更为直观的 SQL 报告输出，辅助分析和诊断。

该函数的主要参数如下图所示：

```
FUNCTION REPORT_SQL_MONITOR RETURNS CLOB
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --------
 SQL_ID                         VARCHAR2                IN     DEFAULT
 SESSION_ID                     NUMBER                  IN     DEFAULT
 SESSION_SERIAL                 NUMBER                  IN     DEFAULT
 SQL_EXEC_START                 DATE                    IN     DEFAULT
 SQL_EXEC_ID                    NUMBER                  IN     DEFAULT
 INST_ID                        NUMBER                  IN     DEFAULT
 START_TIME_FILTER              DATE                    IN     DEFAULT
 END_TIME_FILTER                DATE                    IN     DEFAULT
 INSTANCE_ID_FILTER             NUMBER                  IN     DEFAULT
 PARALLEL_FILTER                VARCHAR2                IN     DEFAULT
 PLAN_LINE_FILTER               NUMBER                  IN     DEFAULT
 EVENT_DETAIL                   VARCHAR2                IN     DEFAULT
 BUCKET_MAX_COUNT               NUMBER                  IN     DEFAULT
 BUCKET_INTERVAL                NUMBER                  IN     DEFAULT
 BASE_PATH                      VARCHAR2                IN     DEFAULT
 LAST_REFRESH_TIME              DATE                    IN     DEFAULT
 REPORT_LEVEL                   VARCHAR2                IN     DEFAULT
 TYPE                           VARCHAR2                IN     DEFAULT
 SQL_PLAN_HASH_VALUE            NUMBER                  IN     DEFAULT
```

通常情况下，提供 SQL_ID 等少数参数，即可生成报告，TYPE 参数用于指定报告类型，这里可以指定生成：TEXT、HTML、XML、ACTIVE 模式的报告。ACTIVE 模式的报告最为华丽直观。

首先可以通过查询 v$sql_monitor 获得那些被监控收集过的 SQL 信息：

```
SQL> select sql_id from v$sql_monitor;


SQL_ID
-------------
6rqxj647ut9pn
f4kcr0dn9rv6z
f6cz4n8y72xdc
```

以下是简单的查询语句，用于生成 HTML 类型的报告：

```
SET LONG 1000000
SET LONGCHUNKSIZE 1000000
```

```
SET LINESIZE 1000
SET PAGESIZE 0
SET TRIM ON
SET TRIMSPOOL ON
SET ECHO OFF
SET FEEDBACK OFF
SELECT DBMS_SQLTUNE.report_sql_monitor(
  sql_id       => '6rqxj647ut9pn',
  type         => 'HTML',
  report_level => 'ALL') AS report
FROM dual;
```

下图是报告的页面展示：

SQL Monitoring Report

SQL Text

select A.object_id , nvl(A.num1, 0) , nvl(A.num2, 0) from (select object_id , count(1) num1,count(decode(is_locked,1,1)) num2 from la_v_invite_bidinfo_gwwz s where exists (select 1 from la_package_detail k where k.material_id = s.material_id) group by object_id union all select xo.object_id , count(1) num1 ,count(1) num2 from la_to_other_object xo where xo.is_locked = 1 and exists (select 1 from la_package_detail k where k.material_id = xo.material_id) group by xo.object_id) A where (nvl(A.num1, 0) = nvl(A.num2, 0))

Global Information: **DONE (ALL ROWS)**

Instance ID	: 1
Session	: BIDPRO1 (191:7)
SQL ID	: 6rqxj647ut9pn
SQL Execution ID	: 16777216
Execution Started	: 09/07/2012 09:38:05
First Refresh Time	: 09/07/2012 09:38:57
Last Refresh Time	: 09/07/2012 09:40:14
Duration	: 129s
Module/Action	: SQL*Plus/-
Service	: SYS$USERS
Program	: sqlplus@enmoteam2 (TNS V1-V3)
Fetch Calls	: 160

Buffer Gets	IO Requests	Database Time	Wait Activity
1M	2196	130s	100%

最全面的报告是 ACTIVE 类型，这个类型的报告会通过 OTN 站点获得展现的框架和 JS 脚本，如果不能连接到公网，你可以在本地构建相应的文件，我在自己的站点保存了这些脚本：

```
mkdir -p eygle.com/sqlmon
cd eygle.com/sqlmon
wget --mirror --no-host-directories --cut-dirs=1 http://download.oracle.com/otn_software/emviewers/scripts/flashver.js
wget --mirror --no-host-directories --cut-dirs=1 http://download.oracle.com/otn_software/emviewers/scripts/loadswf.js
wget --mirror --no-host-directories --cut-dirs=1 http://download.oracle.com/otn_software/emviewers/scripts/document.js
wget --mirror --no-host-directories --cut-dirs=1 http://download.oracle.com/otn_software/emviewers/sqlmonitor/11/sqlmonitor.swf
```

这样在生成 SQL 报告时，就可以调用自己网站的脚本文件。以下是通过脚本调用，生成了一个 ACTIVE 类型的报告：

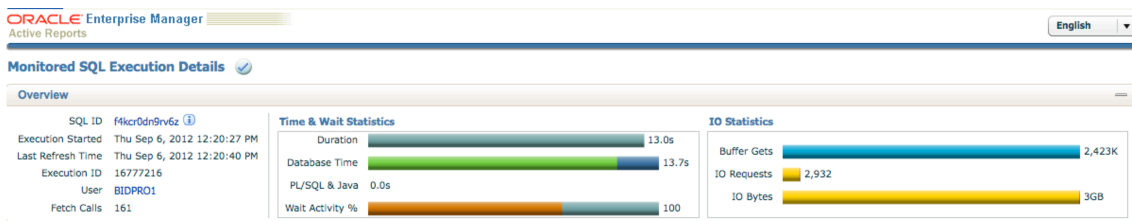
```
[eygle@enmoteam2 ~]$ sqlplus "/ as sysdba" @eygle.sql

SQL*Plus: Release 11.2.0.3.0 Production on Thu Sep 6 15:01:55 2012
```

```
Copyright (c) 1982, 2011, Oracle.  All rights reserved.


Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.3.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
```

在脚本 eygle.sql 中定义了 SQL_ID ,通过这个 SQL_ID 生成了 ACTIVE REPORT：

```
SET LONG 1000000
SET LONGCHUNKSIZE 1000000
SET LINESIZE 1000
SET PAGESIZE 0
SET TRIM ON
SET TRIMSPOOL ON
SET ECHO OFF
SET FEEDBACK OFF

SPOOL report_sql_monitor.htm
SELECT DBMS_SQLTUNE.report_sql_monitor(
  sql_id       => '6rqxj647ut9pn',
  type         => 'ACTIVE',
  report_level => 'ALL',
  base_path    => 'http://www.eygle.com/sqlmon') AS report
FROM dual;
SPOOL OFF
```

这样生成的报告较以前的 SQL Report 更为直观，报告的第一部分展示了 SQL 的执行时间、逻辑读、IO 请求次数及读取数据量等信息：



接下来是具体细节，包括执行计划的 Flash 展现，可以通过柱状图清晰看到各个执行步骤的时间消耗比重，以及 CPU 消耗：

Details

Plan Statistics | Plan | Activity | Metrics

Plan Hash Value　2011087284

TIP: Right mouse click on the table allows to toggle between IO Requests and IO Bytes

Operation	Name	Estimate...	Cost	Timeline(13s)	Exec...	Actual...	Memo...	Temp ...	IO Requests	CPU Activity %	Wait Activity %
SELECT STATEMENT					1	2,395					
FILTER					1	2,395					
HASH JOIN OUTER		5	131K		1	2,561	1MB				
VIEW		5	65K		1	2,399					
UNION-ALL					1	2,399					
FILTER					1	1,716					
HASH GROUP BY		3	44K		1	1,716	1MB				
HASH JOIN SEMI		12K	44K		1	69K	6MB				
HASH JOIN RIGHT ANTI		12K	42K		1	82K	1MB				
TABLE ACCESS FULL	LB_T_PROJECT	374	34		1	374					
VIEW		15K	42K		1	927K				10	
UNION-ALL					1	927K					
NESTED LOOPS OUTER		15K	23K		1	903K					
HASH JOIN RIGHT ANTI		15K	23K		1	903K	2MB				
INDEX FAST FULL SCAN	TEST_IDX	24K	35		1	24K					
HASH JOIN		15K	23K		1	926K	2MB				
INDEX FAST FULL S...	IDX_PUB_ORGAN_CC	23K	26		1	23K					
NESTED LOOPS		15K	23K		1	926K					
HASH JOIN RIGHT...		15K	23K		1	926K	1MB			10	
VIEW	LA_V_MDM_MATERIA	17	19		1	4,658					
HASH JOIN		17	19		1	4,658	1MB				
NESTED LO...		12	9		1	311					

在 PLAN 页面，还有执行计划的图形展现，非常清晰直观：

Plan Statistics | Plan | Activity | Metrics

Graphical　Tabular

对于并行执行的 SQL，还可以通过相应的并行执行页面，显示不同进程的执行性能等：

Details

Plan Statistics | Plan | Parallel | Activity | Metrics

Show Instance Nodes

TIP: Right mouse click on the table allows to toggle between IO Requests and IO Bytes

Parallel Server	Database Time	Wait Activity %	IO Requests	Cell Offload Efficiency	Buffer Gets
All Parallel Servers					
Instance 1					
Parallel Coordinator	0.7s		1		7,317
Parallel Set 1	15.8m	43	145K		903K
Parallel Set 2	3.0m	7.72	9,893		1,222K
Instance 2					
Parallel Set 1	15.6m	42	144K		881K
Parallel Set 2	2.5m	7.07	8,976		1,111K

这是 Oracle 数据库在自动化诊断方面的又一增强，也可以通过 OEM 来观察其输出展现。

9.2.4 从 V$SQLTEXT 中追踪

在数据库出现瓶颈时，通常可以从 V$SESSIION_WAIT 找到那些正在等待资源的 Session，通过 Session 的 sid，联合 V$SESSION 和 V$SQLTEXT 视图可以捕获这些 Session 正在执行的 SQL 语句。

以下是一个生产数据库的问题诊断和解决过程，可以从中体会一下等待事件在解决问题中的指导作用。该生产环境的操作系统为 Solaris 8，数据库版本为 9.1.7.4，业务及开发人员报告系统运行缓慢，已经影响业务系统正常使用，请求协助诊断。

数据库运行缓慢，转换为数据库语言那就是数据库可能经历了等待，那么可以通过 V$SESSION_WAIT（从 Oracle 10g，V$SESSION 视图可以取代 V$SESSION_WAIT 的这一诊断功能）视图来入手。查询 V$SESSION_WAIT 获取各进程等待事件：

```
SQL> select sid,event,p1,p1text from v$session_wait;
       SID EVENT                              P1 P1TEXT
---------- ------------------------------ ---------- ------------------------------
       124 latch free                     1.6144E+10 address
       140 buffer busy waits                      17 file#
        66 buffer busy waits                      17 file#
        10 db file sequential read                17 file#
        18 db file sequential read                17 file#
        54 db file sequential read                17 file#
        49 db file sequential read                17 file#
        48 db file sequential read                17 file#
        46 db file sequential read                17 file#
        45 db file sequential read                17 file#
……
244 rows selected.
```

对于本案例，通过以上输出发现存在大量 db file scattered read 及 db file sequential read 等待，并且全表扫描的等待都位于文件号为 17 的数据文件上。显然全表扫描等操作成为系统最严重的性能影响因素。

> 说明：db file scattered read（DB 文件分散读取）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，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。

9.2.5 捕获相关 SQL

确定这些进程因为数据访问产生了等待，可以考虑捕获这些 SQL 以发现问题。这里用到了以下脚本 getsqlbysid.sql，该脚本通过已知 session 的 sid，联合 v$session、v$sqltext 视图，获得相关 Session 正在执行的完整 SQL 语句。

```
SELECT   sql_text  FROM v$sqltext a
   WHERE a.hash_value = (SELECT sql_hash_value FROM v$session b WHERE b.SID = '&sid')
ORDER BY piece ASC;
```

使用该脚本，通过从 v$session_wait 中获得的等待全表或索引扫描的进程 SID，捕获问题 SQL：

```
SQL> @getsqlbysid
Enter value for sid: 18
old   5: where b.sid='&sid'
new   5: where b.sid='18'
SQL_TEXT
----------------------------------------------------------------------------------
select i.vc2title,i.numinfoguid  from  hs_info i where i.intenabledflag = 1
 and i.intpublishstate = 1  and i.datpublishdate <=sysdate  and i.numcatalogguid = 2047
order by i.datpublishdate desc, i.numorder desc
SQL> /
Enter value for sid: 54
old   5: where b.sid='&sid'
new   5: where b.sid='54'
SQL_TEXT
----------------------------------------------------------------------------------
select i.vc2title,i.numinfoguid  from  hs_info i where i.intenabledflag = 1
and i.intpublishstate = 1  and i.datpublishdate <=sysdate  and i.numcatalogguid = 33
order by i.datpublishdate desc, i.numorder desc
SQL> /
Enter value for sid: 49
old   5: where b.sid='&sid'
new   5: where b.sid='49'
SQL_TEXT
------------------------------------------------------------------------------------
select i.vc2title,i.numinfoguid  from  hs_info i where i.intenabledflag = 1
and i.intpublishstate = 1  and i.datpublishdate <=sysdate  and i.numcatalogguid = 26
order by i.datpublishdate desc, i.numorder desc
```

对几个进程进行跟踪，分别得到以上 SQL 语句，这些 SQL 可能就是问题产生的根源。以上语句如果良好编码应该使用绑定变量，但是目前这个不是我们关心的。

使用该应用用户连接，通过 SQL*Plus 的 AUTOTRACE 功能检查以上 SQL 的执行计划：

```
SQL> set autotrace trace explain
SQL> select i.vc2title,i.numinfoguid
  2  from  hs_info i where i.intenabledflag = 1
  3  and i.intpublishstate = 1  and i.datpublishdate <=sysdate
  4  and i.numcatalogguid = 3475
  5  order by i.datpublishdate desc, i.numorder desc  ;
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=228 Card=1 Bytes=106)
   1    0   SORT (ORDER BY) (Cost=228 Card=1 Bytes=106)
   2    1     TABLE ACCESS (FULL) OF 'HS_INFO' (Cost=218 Card=1 Bytes=106)
SQL> select count(*) from hs_info;
  COUNT(*)
----------
    227404
```

通过执行计划看到以上查询使用了全表扫描，而该表这里有 22 万记录，全表扫描已经不再适合。检查该表，存在以下索引：

```
SQL> select index_name,index_type from user_indexes where table_name='HS_INFO';
INDEX_NAME                     INDEX_TYPE
------------------------------ ---------------------------
HSIDX_INFO1                    FUNCTION-BASED NORMAL
HSIDX_INFO_SEARCHKEY        DOMAIN
PK_HS_INFO                     NORMAL
```

进一步的检查该表索引键值：

```
SQL> select index_name,column_name from user_ind_columns where table_name ='HS_INFO';
INDEX_NAME                     COLUMN_NAME
------------------------------ --------------------
HSIDX_INFO1                    NUMORDER
HSIDX_INFO1                    SYS_NC00024$
HSIDX_INFO_SEARCHKEY           VC2INDEXWORDS
PK_HS_INFO                     NUMINFOGUID
SQL> desc hs_info
 Name                                Null?    Type
 ----------------------------------- -------- --------------------------------------------
 NUMINFOGUID                         NOT NULL NUMBER(15)
 NUMCATALOGGUID                      NOT NULL NUMBER(15)
 INTTEXTTYPE                         NOT NULL NUMBER(38)
 VC2TITLE                            NOT NULL VARCHAR2(60)
```

```
 VC2AUTHOR                                   VARCHAR2(100)
……
```

检查发现在 numcataloggguid 字段上并没有索引，该字段具有很好的区分度，考虑在该字段创建索引以消除全表扫描。

```
SQL> create index hs_info_NUMCATALOGGUID on hs_info(NUMCATALOGGUID);
Index created.
SQL> set autotrace trace explain
SQL> select i.vc2title,i.numinfoguid
  2  from  hs_info i where i.intenabledflag = 1
  3  and i.intpublishstate = 1  and i.datpublishdate <=sysdate
  4  and i.numcatalogguid = 3475
  5  order by i.datpublishdate desc, i.numorder desc ;
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=12 Card=1 Bytes=106)
   1    0   SORT (ORDER BY) (Cost=12 Card=1 Bytes=106)
   2    1     TABLE ACCESS (BY INDEX ROWID) OF 'HS_INFO' (Cost=2 Card=1 Bytes=106)
   3    2       INDEX (RANGE SCAN) OF 'HS_INFO_NUMCATALOGGUID' (NON-UNIQUE) (Cost=1
Card=1)
```

观察系统状况，原大量等待消失：

```
SQL> select sid,event,p1,p1text from v$session_wait where event not like 'SQL%';
       SID EVENT                                     P1 P1TEXT
---------- ------------------------------ ---------- -------------------------
         1 pmon timer                               300 duration
         2 rdbms ipc message                        300 timeout
         3 rdbms ipc message                        300 timeout
         6 rdbms ipc message                     180000 timeout
        59 rdbms ipc message                       6000 timeout
       118 rdbms ipc message                       6000 timeout
       275 rdbms ipc message                      30000 timeout
       147 rdbms ipc message                       6000 timeout
        62 rdbms ipc message                       6000 timeout
        11 rdbms ipc message                      30000 timeout
         4 rdbms ipc message                        300 timeout
       305 db file sequential read                   17 file#
       356 db file sequential read                   17 file#
        19 db file scattered read                    17 file#
         5 smon timer                              300 sleep time
15 rows selected.
```

至此，此问题得以解决。

通过以上案例，可以知道从等待事件进行追踪的诊断方法，这种方法在日常数据库诊断中很常用，在后面章节中还将会进一步的详细介绍。

9.3 Oracle 10g 的增强

虽然 V$SESSION_WAIT 记录的信息如此重要，但是这些重要的信息是随 Session 而消逝的，如果我们希望获得数据库的历史状态及 Session 的历史等待信息等数据，是不可得的。

所以很多时候很难回答这样的问题：

- 这个系统昨天是什么样子的？
- 今天和昨天相比有什么不同？
- 1 个小时前的那次性能下滑是哪个用户引起的？
- 是哪些事件使我们今天用了更多的时间来等待？

你也可能一次又一次地听到 Oracle Support 这样问：

- 问题出现时系统是怎样的状况？
- 问题出现时系统有哪些等待？
- 你能否重现（Reproduce）问题以便我们判断？

很多这样的问题是极其使人恼火的，我们当然不希望问题重现（reproduce）再次引起宕机或业务损失，而那些问题看起来分明是不作为的责任推卸。可是事实是，失去了现场和当时的状态以及 Session 的实时信息，DBA 也的确很难判断问题的所在。

从 Oracle 10g 开始，Oracle 开始改变这一切，所以赘述这么多，我只想更郑重地告诉大家，这一改变是多么的重要。

9.3.1 新增 v$session_wait_history 视图

为了更有效地保留 Session 信息，Oracle 10g 新增加了一个 v$session_wait_history 视图，该视图用以记录活动 Session 的最近 10 次等待事件。以下查询输出了 SID 为 120 的会话最近的 10 次等待，注意其中关于 db file sequential read 等待事件的记录，可以从中得知每次等待发生的文件号以及数据块，以前这些信息一旦成为历史就无法获取：

```
SQL> select event,p1text,p1,p2text,p2,p3text,p3,wait_time
  2  from v$session_wait_history where sid=120;
EVENT                         P1TEXT    P1 P2TEXT     P2 P3TEXT   P3  WAIT_TIME
---------------------------- -------- --- ------- ------ -------- -- ----------
db file sequential read      file#     14 block#   97456 blocks    1          0
row cache lock               cache id  11 mode         0 request   3         49
row cache lock               cache id  11 mode         0 request   3          0
db file sequential read      file#     10 block#  260171 blocks    1          1
db file sequential read      file#     14 block#  570536 blocks    1         10
```

```
db file sequential read      file#     14 block#    6363 blocks    1         12
db file sequential read      file#     14 block#   35285 blocks    1          9
db file sequential read      file#     14 block#   40674 blocks    1          9
db file sequential read      file#     14 block#   69631 blocks    1          1
db file sequential read      file#     14 block#   82498 blocks    1          3
10 rows selected
```

v$session_wait_history 缺省记录活动会话最近的 10 次等待，这个约束受到一个隐含参数的影响，这个参数就是_session_wait_history，其缺省值是 10，如果想保留活动会话更多的等待，可以通过修改这个隐含参数来进行：

```
SQL> @GetHidPar
Enter value for par: session_wait
old   4:    AND x.ksppinm LIKE '%&par%'
new   4:    AND x.ksppinm LIKE '%session_wait%'
NAME                           VALUE      PDESC
------------------------------ ---------- ------------------------------------------
_session_wait_history          10         enable session wait history collection
```

通过 v$session_wait_history 这个视图，可以将 V$SESSION_WAIT 的功能进行延伸，获取更多的相关信息辅助数据库问题诊断。这是 Oracle 迈出的一小步。

9.3.2 ASH 新特性

如果说 v$session_wait_history 是一小步，那么 ASH 则是 Oracle 迈出根本变革的一大步。

从 Oracle 10g 开始，Oracle 引入了 ASH 新特性，也就是活动 Session 历史信息记录（Active Session History，ASH）。ASH 以 V$SESSION 为基础，每秒钟采样一次，记录活动会话等待的事件。因为记录所有会话的活动是非常昂贵的，所以不活动的会话不会被采样，这一点从 ASH 的“A”上就可以看出。采样工作由 Oracle 10g 新引入的一个后台进程 MMNL 来完成。

是否启用 ASH 功能，受一个内部隐含参数控制：

```
SQL> @GetHparDes.sql
Enter value for par: ash_en
old   6:    AND x.ksppinm LIKE '%&par%'
new   6:    AND x.ksppinm LIKE '%ash_en%'
NAME          VALUE    DESCRIB
------------- -------- ---------------------------------------------------------
……
_ash_enable    TRUE     To enable or disable Active Session sampling and flushing
```

而采样时间同样由另一个内部隐含参数决定：

```
SQL> @GetHparDes.sql
Enter value for par: ash_sampling
old   6:    AND x.ksppinm LIKE '%&par%'
```

```
new   6:    AND x.ksppinm LIKE '%ash_sampling%'
NAME                   VALUE  DESCRIB
---------------------- ------ -----------------------------------------------------------
_ash_sampling_interval 1000   Time interval between two successive Active Session samples in
                              millisecs
```

1000 毫秒，正好是 1 秒的时间。

> 注意：隐含参数通常具有特殊的作用，一般不建议用户查询或者修改，本文大量引用隐含参数的目的只有一个，那就是希望大家知道，所有我们在文档中见到的限制、约束、阈值、比率都是有来源的，只要足够细心，我们就能找出数据库的真相，不再靠记忆来学习。

很多人可能更关心性能，如此频繁的采样是否会极大地影响数据库的性能呢？采样的性能影响无疑是存在的，但是因为 Oracle 的采样工具可以直接访问 Oracle 10g 内部结构，所以是极其高效的，对于性能的影响也非常小，这也正是 Oracle 提供优化或诊断工具的优势所在。

ASH 信息被设计为在内存中滚动的，在需要的时候早期的信息是会被覆盖的。ASH 记录的信息可以通过 v$active_session_history 视图来访问，对于每个活动 SESSION，每次采样会在这个视图中记录一行信息。

这部分内存在 SGA 中分配：

```
SQL> select * from v$sgastat where name like '%ASH%';
POOL            NAME                          BYTES
------------    ----------------------        ----------
shared pool     ASH buffers                   6291456
```

注意 ASH buffers 的最小值为 1MB，最大值不超过 30MB，大小按照以下算法分配：

*Max (Min (cpu_count * 2MB, 5% * SHARED_POOL_SIZE, 30MB), 1MB)*

在以上公式中，如果 SHARED_POOL_SIZE 未显示设置，则限制为 2%*SGA_TARGET。这一算法在 Oracle 10g 的不同版本中，可能不同。根据这个算法，我的采样系统分配的 ASH Buffers 为 6MB。

```
SQL> select name,value,display_value from v$parameter
  2  where name in ('shared_pool_size','cpu_count');
NAME                         VALUE                DISPLAY_VALUE
---------------------------- -------------------- --------------------
cpu_count                    4                    4
shared_pool_size             125829120            120M
```

另外一个生产系统中，这一内存分配为 8MB：

```
SQL> select * from v$version where rownum <2;
BANNER
----------------------------------------------------------------
Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Prod
SQL> show parameter cpu_count
NAME                                 TYPE        VALUE
```

```
---------------------------------- ---------- -----------------------------
cpu_count                          integer    4
SQL> show parameter sga_target
NAME                               TYPE       VALUE
---------------------------------- ---------- -----------------------------
sga_target                         big integer 900M
SQL> show parameter shared_pool_size
NAME                               TYPE       VALUE
---------------------------------- ---------- -----------------------------
shared_pool_size                   big integer 0
SQL> select * from v$sgastat where name like 'ASH%';
POOL         NAME                             BYTES
------------ ---------------------------- ----------
shared pool  ASH buffers                     8388608
```

记录在 SGA 中的 ASH 信息，可以通过 v$session_wait_history 进行查询：

```
SQL> desc v$session_wait_history
Name       Type         Nullable Default Comments
---------- ------------ -------- ------- --------
SID        NUMBER         Y
SEQ#       NUMBER         Y
EVENT#     NUMBER        Y
EVENT      VARCHAR2(64)   Y
P1TEXT     VARCHAR2(64)   Y
P1         NUMBER         Y
P2TEXT     VARCHAR2(64)   Y
P2         NUMBER         Y
P3TEXT     VARCHAR2(64)   Y
P3         NUMBER         Y
WAIT_TIME  NUMBER        Y
WAIT_COUNT NUMBER       Y
```

可以通过 Oracle 提供的工具生成 ASH 的报告，报告可以以几分钟未为跨度对数据库进行精确分析；也可以以数小时或数天为时间跨度，为数据库提供概要分析。

生成 ASH 报告主要可以通过两种方式：脚本方式和 OEM 图形方式。

1．脚本方式

调用$ORACLE_HOME/rdbms/admin/ashrpt.sql 脚本，回答一系列问题之后，就可以生成一个 ASH 的报告，报告包括 TOP 等待事件、TOP SQL、TOP SQL 命令类型、TOP Sessions 等内容，摘录部分报告内容如下。

调用 ashrpt.sql 脚本：

```
SQL> @?/rdbms/admin/ashrpt.sql
Current Instance
~~~~~~~~~~~~~~~~
   DB Id    DB Name      Inst Num Instance
----------- ------------ -------- ------------
 3965153484 DANALY              1 danaly
……
ASH Samples in this Workload Repository schema
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

数据库可用的采样数据：

```
Oldest ASH sample available:  31-Mar-06 08:31:52   [   4325 mins in the past]
Latest ASH sample available:  04-Sep-06 22:39:11   [ ###### mins in the past]
……
```

用户定义概要如下：

```
Summary of All User Input
-------------------------
Format         : TEXT
DB Id          : 3965153484
Inst num       : 1
Begin time     : 02-Apr-06 08:37:42
End time       : 03-Apr-06 08:37:59
Slot width     : Default
Report targets : 0
Report name    : ashrpt_1_0403_0837.txt
```

生成的报告如下：

```
ASH Report For DANALY/danaly

DB Name         DB Id    Instance     Inst Num Release     RAC Host
------------ ----------- ------------ -------- ----------- --- ------------
DANALY        3965153484 danaly              1 10.2.0.1.0  NO  danaly.hurrr

CPUs           SGA Size       Buffer Cache        Shared Pool    ASH Buffer Size
---- ------------------ ------------------ ------------------ ------------------
   4        900M (100%)       772M (85.8%)       210M (23.3%)        9.0M (0.9%)

          Analysis Begin Time:   02-Apr-06 08:37:42
            Analysis End Time:   03-Apr-06 08:37:59
                 Elapsed Time:   1,440.3 (mins)
                 Sample Count:   2,946
```

```
      Average Active Sessions:     0.03
  Avg. Active Session per CPU:     0.01
               Report Target:   None specified

Top User Events               DB/Inst: DANALY/danaly  (Apr 02 08:37 to 08:37)
                                                               Avg Active
Event                               Event Class     % Activity   Sessions
----------------------------------- --------------- ---------- ----------
CPU + Wait for CPU                  CPU                  22.84       0.01
log file sync                       Commit               19.23       0.01
          -------------------------------------------------------------

Top Background Events         DB/Inst: DANALY/danaly  (Apr 02 08:37 to 08:37)

                                                               Avg Active
Event                               Event Class     % Activity   Sessions
----------------------------------- --------------- ---------- ----------
log file parallel write             System I/O           21.83       0.01
control file parallel write         System I/O           15.44       0.01
db file parallel write              System I/O           15.41       0.01
CPU + Wait for CPU                  CPU                   5.26       0.00
          -------------------------------------------------------------
……..

Top SQL Command Types         DB/Inst: DANALY/danaly  (Apr 02 08:37 to 08:37)
……..
                                           Distinct            Avg Active
SQL Command Type                             SQLIDs % Activity   Sessions
---------------------------------------- ---------- ---------- ----------
INSERT                                            8      11.30       0.00
SELECT                                           54       6.79       0.00
PL/SQL EXECUTE                                   21       2.17       0.00
UPDATE                                           10       2.07       0.00
          -------------------------------------------------------------

Top SQL Statements            DB/Inst: DANALY/danaly  (Apr 02 08:37 to 08:37)

       SQL ID    Planhash % Activity Event                             % Event
------------- ----------- ---------- ------------------------------ ----------
```

```
74y62ap82k1xk  2315018254      7.74 CPU + Wait for CPU                 7.74
INSERT INTO MGMT_CURRENT_METRICS (TARGET_GUID, KEY_VALUE, COLLECTION_TIMESTAMP,
METRIC_GUID, VALUE, STRING_VALUE) VALUES (:B1 , :B2 , :B3 , :B4 , :B5 , :B6 )
……

          -------------------------------------------------------------
Top Sessions                   DB/Inst: DANALY/danaly  (Apr 02 08:37 to 08:37)
……

End of Report
Report written to ashrpt_1_0403_0837.txt
```

2．OEM 图形方式

使用 OEM，可以在性能页，单击“运行 ASH 报告”按钮生成 ASH 报告，如图 9-1 所示。

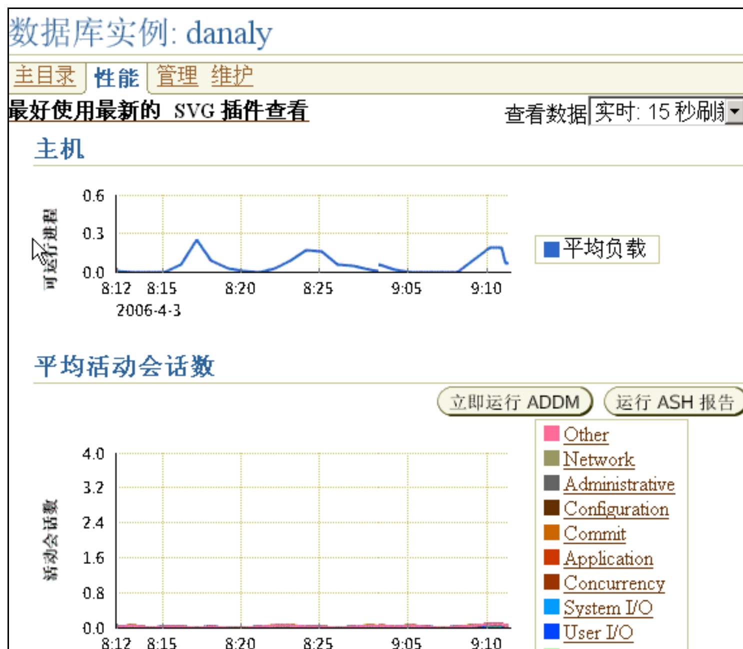


图 9-1　生成 ASH 报告

OEM 生成的 ASH 报告非常清晰和直观。ASH 的概况信息如图 9-2 所示。等待事件信息如图 9-3 所示，等待参数信息如图 9-4 所示，TOP SQL 信息如图 9-5 所示。只要试用一下就可以感受到 ASH 的强大功能。

DB Name	DB Id	Instance	Inst num	Release	RAC	Host
DANALY	3965153484	danaly	1	10.2.0.1.0	NO	danaly.hurrray.com.cn

CPUs	SGA Size	Buffer Cache	Shared Pool	ASH Buffer Size
4	900M (100%)	768M (85.3%)	214M (23.7%)	8.0M (0.9%)

	Sample Time	Data Source
Analysis Begin Time:	31-Mar-06 21:01:20	V$ACTIVE_SESSION_HISTORY
Analysis End Time:	31-Mar-06 22:06:20	V$ACTIVE_SESSION_HISTORY
Elapsed Time:	65.0 (mins)	
Sample Count:	2,421	
Average Active Sessions:	0.62	
Avg. Active Session per CPU:	0.16	
Report Target:	None specified	

图 9-2　ASH 概况信息

Top User Events

Event	Event Class	% Activity	Avg Active Sessions
db file scattered read	User I/O	27.05	0.17
SQL*Net more data from dblink	Network	9.25	0.06
CPU + Wait for CPU	CPU	8.55	0.05
db file sequential read	User I/O	3.97	0.02
direct path read temp	User I/O	3.39	0.02

Back to Top Events
Back to Top

Top Background Events

Event	Event Class	% Activity	Avg Active Sessions
db file parallel write	System I/O	24.20	0.15
log file parallel write	System I/O	14.21	0.09
control file parallel write	System I/O	1.57	0.01
CPU + Wait for CPU	CPU	1.03	0.01

图 9-3　等待事件信息

Top Event P1/P2/P3 Values

Event	% Event	P1 Value, P2 Value, P3 Value	% Activity	Parameter 1	Parameter 2	Parameter 3
db file scattered read	27.05	"10","926205","16"	0.08	file#	block#	blocks
db file parallel write	24.20	"1","0","2147483647"	13.96	requests	interrupt	timeout
		"2","0","2147483647"	3.39			
		"3","0","2147483647"	1.53			
log file parallel write	14.21	"2","26","2"	1.40	files	blocks	requests
SQL*Net more data from dblink	9.25	"675562835","1","0"	2.81	driver id	#bytes	NOT DEFINED
		"675562835","11","0"	2.73			
		"675562835","5","0"	2.11			
db file sequential read	4.05	"3","80812","1"	0.04	file#	block#	blocks

图 9-4　等待参数信息

Top SQL Statements

SQL ID	Planhash	% Activity	Event	% Event	SQL Text
gz9x4u45c91mb	1976783940	22.02	SQL*Net more data from dblink	9.00	DELETE FROM CM_TB_MMS_DATA_NEW...
			db file scattered read	6.90	
			direct path read temp	3.22	
5p1nu5zz3d5vq	2780303235	11.90	db file scattered read	9.38	delete from CM_TB_MMS_DATA_OLD...
			db file sequential read	1.28	
			CPU + Wait for CPU	1.24	
3zmwscncd42zg	2489428218	4.63	db file scattered read	4.13	select count(*) from CM_TB_MMS...
gs6rw17g1zbsn	1954154723	3.30	db file scattered read	2.48	delete from CM_TB_MMS_DATA_NEW...
7aksy1t0qms5s	4060839695	2.44	db file scattered read	1.32	DELETE FROM CM_TB_MMS_DATA_NEW...
			CPU + Wait for CPU	1.07	

图 9-5　TOP SQL 信息

只要试用一下就可以感受到 ASH 的强大功能。

9.3.3 自动负载信息库：AWR 的引入

内存中记录的 ASH 信息始终是有限的，为了保存历史数据，这些信息最终需要写入磁盘。这些历史信息的存储，引出了 Oracle10g 的另外一个新特性：自动负载信息库（Automatic Workload Repository，AWR）。

1．AWR 的采样机制

AWR 收集关于该特定数据库的操作统计信息和其他统计信息，Oracle 以固定的时间间隔（默认为每小时一次）为其所有重要统计信息和负载信息执行一次快照，并将这些快照存储在 AWR 中。这些信息在 AWR 中保留给定的时间（默认为一周），然后被清除。执行快照的频率及其保持时间都可以自定义，以满足不同环境的独特需要。

AWR 的采样工作由后台进程 MMON 每 60 分钟执行一次，ASH 信息同样会被采样写出到 AWR 负载库。虽然 ASH buffers 被设计为保留 1 小时的信息，但是很多时候这个内存是不

足够的，当 ASH buffers 写满之后，另外一个后台进程 MMNL 将会主动将 ASH 信息写出。由于数据量巨大，把所有的 ASH 数据写到磁盘上是不可接受的。一般是在写到磁盘的时候过滤这个数据，写出的数据占采样数据的 10%，写出时通过 direct-path insert 完成，尽量减少日志生成，从而最小化数据库性能影响。

通过图 9-6 可以理解一下 ASH 与 AWR 的关系。

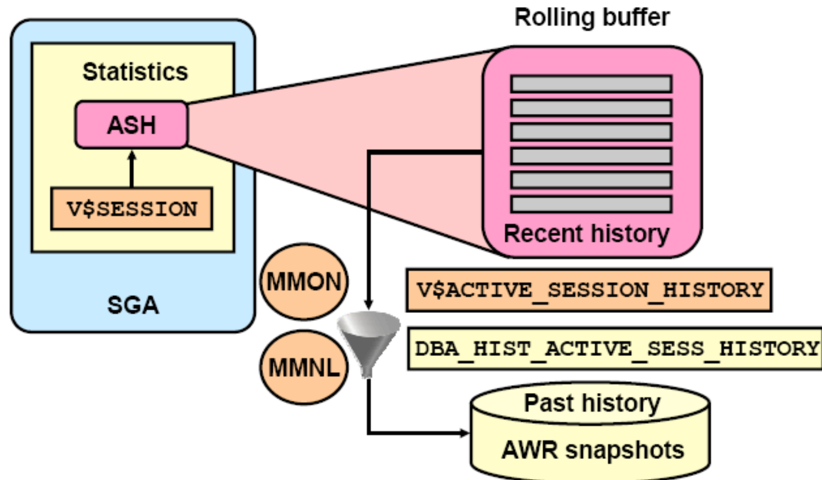


图 9-6　ASH 与 AWR 的关系

AWR 的行为受到数据库另外一个重要初始化参数 STATISTICS_LEVEL 的影响，该参数有 3 个可选值。

- BASIC：设置为 BASIC 时，AWR 的统计信息收集和所有自我调整的特性都被关闭。
- TYPICAL：设置为 TYPICAL 时，数据库收集部分统计信息，这些信息为典型的数据库监控需要，是数据库的缺省设置。
- ALL：所有可能的统计信息都被收集。

ASH 信息的写出比例受一个隐含参数控制：

```
SQL> @GetHparDes.sql
Enter value for par: filter_ratio
old   6:    AND x.ksppinm LIKE '%&par%'
new   6:    AND x.ksppinm LIKE '%filter_ratio%'
NAME                           VALUE      DESCRIB
------------------------------ ---------- ---------------------------------------------
_ash_disk_filter_ratio         10         Ratio of the number of in-memory samples to the
                                          number of samples actually written to disk
```

写出到 AWR 负载库的 ASH 信息记录在 AWR 的基础表 wrh$active_session_hist 中，wrh$active_session_hist 是一个分区表，Oracle 会自动进行数据清理。

wrh$active_session_hist 记录的这些历史信息，可以通过 dba_hist_active_sess_history 视图进行聚合查询，通过简化后的图 9-7 来看一下 Oracle 以 Session 为起点的一系列用以追踪和诊断的数据库对象。

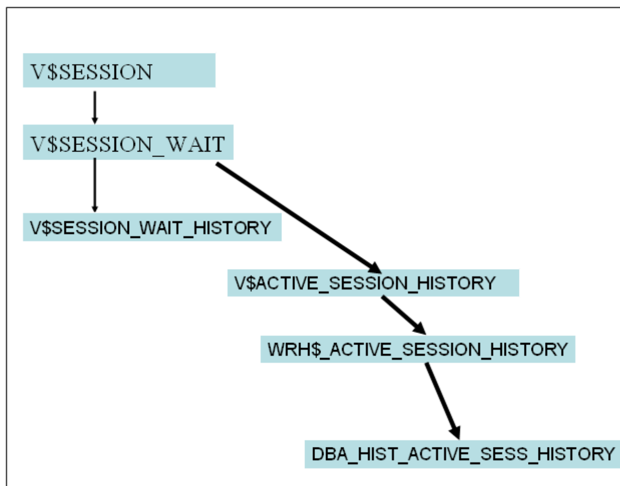


图 9-7　一系列用以追踪和诊断的数据库对象

简单总结一下：

- V$SESSION 代表数据库活动的开始，是为源起；
- V$SESSION_WAIT 视图用以实时记录活动 SESSION 的等待情况，是当前信息；
- V$SESSION_WAIT_HISTORY 是对 V$SESSION_WAIT 的简单增强，记录活动 SESSION 的最近 10 次等待；
- V$ACTIVE_SESSION_HISTORY 是 ASH 的核心，用以记录活动 Session 的历史等待信息，每秒采样 1 次，这部分内容记录在内存中，期望值是记录 1 个小时的内容；
- WRH$_ACTIVE_SESSION_HISTORY 是 V$ACTIVE_SESSION_HISTORY 在 AWR 的存储地，V$ACTIVE_SESSION_HISTORY 中记录的信息会被定期（每小时 1 次）地刷新到负载库中，并缺省保留一个星期用于分析；
- DBA_HIST_ACTIVE_SESS_HISTORY 视图是 WRH$_ACTIVE_SESSION_HISTORY 视图和其他几个视图的联合展现，我们通常通过这个视图进行历史数据的访问。

通过以上分析过程可以看到，关于 Session 信息的记录，Oracle 从不同的粒度进行了增强，目的只有一个，那就是全面真实地记录、监控和反映数据库的运行状况。

2. AWR 的采样数据存储

AWR 记录的信息还远不止于此，通过系统的自动采样，AWR 可以收集数据库运行的各方面统计信息及等待等重要数据，提供给数据库诊断分析使用。当然 AWR 的信息需要独立存储，在 Oracle 10g 中，新增的 SYSAUX 表空间是这类信息的存储地：

```
SQL> select OCCUPANT_NAME,OCCUPANT_DESC,SCHEMA_NAME,SPACE_USAGE_KBYTES/1024 "MB"
  2  from V$SYSAUX_OCCUPANTS WHERE OCCUPANT_NAME LIKE '%AWR%';
OCCUPANT_NAME OCCUPANT_DESC                                          SCHEMA_NAME      MB
------------- ---------------------------------------------- ------------   --------
SM/AWR     Server Manageability - Automatic Workload Repository      SYS        250.875
```

在 Oracle 10g 之前的版本中，类似的功能是由 Statspack 实现，但是 Statspack 需要由用户自行安装调度，并且其收集的信息十分有限。我们一直提到的 Session 历史信息 Statspack 就是

无法提供的。AWR 大大强化了这部分信息，由于 AWR 收集的信息十分完备，所以经常被称为“数据库的数据仓库”。

AWR 收集的信息通过一系列的视图展现出来，可以查询这些视图获得数据库的信息采样：

```
SQL> select object_name,object_type from dba_objects
  2  where object_name like 'DBA_HIST%' and object_type='VIEW' and rownum <5;

OBJECT_NAME                    OBJECT_TYPE
------------------------------ -------------------
DBA_HIST_DATABASE_INSTANCE     VIEW
DBA_HIST_SNAPSHOT              VIEW
DBA_HIST_SNAP_ERROR            VIEW
DBA_HIST_BASELINE              VIEW
```

这些系统视图的底层表大致有 3 类 WRM$表存储 AWR 的元数据（Workload Repository Metadata），WRH$表存储采样快照的历史数据（Workload Repository Historical），WRI$表存储同数据库建议功能相关的数据。Oracle 10g 中相关表的数量大致如下：

```
SQL> select * from v$version where rownum <2;
BANNER
----------------------------------------------------------------
Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Prod
SQL> select substr(table_name,1,4),count(*) from dba_tables
  2  where table_name like 'WR%' group by substr(table_name,1,4);
SUBSTR(T   COUNT(*)
-------- ----------
WRM$              5
WRH$             94
WRI$             61
```

从 Oracle 11g 开始，这个家族又增加了新的成员，WRR$类表代表的是 Oracle 11g 新功能 Workload Capture 以及 Workload Replay 相关信息：

```
SQL> select substr(table_name,1,4),count(*) from dba_tables
  2  where table_name like 'WR%' group by substr(table_name,1,4);
SUBSTR(T   COUNT(*)
-------- ----------
WRM$              8
WRR$              9
WRH$            113
WRI$             84
SQL> select table_name from dba_tables where table_name like 'WRR%';
TABLE_NAME
------------------------------
```

```
WRR$_REPLAY_STATS
WRR$_REPLAY_SEQ_DATA
WRR$_REPLAY_SCN_ORDER
WRR$_REPLAY_DIVERGENCE
WRR$_REPLAYS
WRR$_FILTERS
WRR$_CONNECTION_MAP
WRR$_CAPTURE_STATS
WRR$_CAPTURES

9 rows selected.
```

AWR 的历史数据表主要通过分区表进行存储，这些分区表信息可以通过 DBA_TAB_PARTITIONS 视图进行查询：

```
SQL> select TABLE_NAME,PARTITION_NAME,TABLESPACE_NAME from dba_tab_partitions
  2  where table_name like 'WR%' and rownum <5;

TABLE_NAME                     PARTITION_NAME                 TABLESPACE_NAME
------------------------------ ------------------------------ -----------------
WRH$_EVENT_HISTOGRAM           WRH$_EVENT__1478080230_347     SYSAUX
WRH$_SYSTEM_EVENT              WRH$_SYSTEM_1478080230_251     SYSAUX
WRH$_SQLSTAT                   WRH$_SQLSTA_1478080230_251     SYSAUX
WRH$_FILESTATXS                WRH$_FILEST_1478080230_251     SYSAUX
```

3．AWR 报告展现

AWR 记录的数据可以通过报告来展现，报告可以通过运行脚本生成类似 Statspack report 的 AWR 报告，生成报告的脚本位于$ORACLE_HOME/rdbms/admin/awrrpt.sql，报表可以通过两种形式输出：TEXT 和 HTML。用脚本生成 AWR 报告的过程与生成 Statspack 报告非常类似，需要以 sys 用户执行这个脚本，执行过程需要选择报表类型、天数（用来决定显示那几天内的 snapshot）、begin_snap、end_snap 以及报表名称等 5 个参数。

以下是一个 HTML 格式报表的展现示例，如图 9-8 所示。

WORKLOAD REPOSITORY report for

DB Name	DB Id	Instance	Inst num	Release	RAC	Host
GHCCDB	1992983269	ghccdb	1	10.2.0.3.0	NO	ccdb2

	Snap Id	Snap Time	Sessions	Cursors/Session
Begin Snap:	4652	17-Jun-08 13:00:50	325	.6
End Snap:	4653	17-Jun-08 14:00:01	329	.6
Elapsed:		59.20 (mins)		
DB Time:		108.83 (mins)		

Report Summary

Cache Sizes

	Begin	End		
Buffer Cache:	960M	960M	Std Block Size:	8K
Shared Pool Size:	1,488M	1,488M	Log Buffer:	14,392K

图 9-8　一个 HTML 格式报表

值得注意的是，从 Oracle 11g 开始，AWR 报告中增加了很多和操作系统相关的信息，这些信息此前无法通过报告获取。新增的内容包括主机 CPU 和内存信息：

```
Host Name        Platform                         CPUs Cores Sockets Memory(GB)
---------------- -------------------------------- ---- ----- ------- ----------
wapdb.eygle.com Linux IA (32-bit)                   2     1       1       1.98
```

负载概要信息增加了 CPU 信息：

```
Load Profile              Per Second    Per Transaction   Per Exec    Per Call
~~~~~~~~~~~~          ---------------   --------------- ---------- ----------
      DB Time(s):                0.0                0.1       0.01       0.00
       DB CPU(s):                0.0                0.1       0.00       0.00
       Redo size:              486.7            6,879.0
……
W/A MB processed:          283,089.4        4,001,459.7
```

以及 CPU 负载信息、实例 CPU 耗用以及内存使用等信息：

```
Host CPU (CPUs:     2 Cores:      1 Sockets:     1)
~~~~~~~~         Load Average
               Begin       End     %User    %System      %WIO     %Idle
           --------- --------- --------- --------- --------- ---------
                0.12      0.06       0.4       0.4       1.6      99.3
Instance CPU
~~~~~~~~~~~~
```

```
              % of total CPU for Instance:   0.4
              % of busy  CPU for Instance:  59.6
  %DB time waiting for CPU - Resource Mgr:   0.0
Memory Statistics
~~~~~~~~~~~~~~~~~                       Begin          End
                  Host Mem (MB):      2,027.1      2,027.1
                   SGA use (MB):        600.0        600.0
                   PGA use (MB):        282.1        283.0
    % Host Mem used for SGA+PGA:        43.52        43.52
```

4．AWR 比较报告诊断案例

值得一提的是 AWR 报告还有另外一种形式的展现，那就是 AWR 比较报告。通常生成 AWR 报告的脚本是 awrrpt.sql，而比较报告可以通过 awrddrpt.sql 生成（这个脚本通过调用 awrddrpi.sql 脚本生成报告）。这个脚本生成报告的过程与 awrrpt.sql 有所不同。

运行这个脚本，可以选择以 HTML 格式生成报告：

```
SQL> @?/rdbms/admin/awrddrpt.sql

Specify the Report Type
~~~~~~~~~~~~~~~~~~~~~~~
Would you like an HTML report, or a plain text report?
Enter 'html' for an HTML report, or 'text' for plain text
Defaults to 'html'
Enter value for report_type:

Type Specified:                  html
```

接下来列出数据库的 DBID 等信息，以下输出来自一个双机热备环境，同一数据库在不同阶段可能运行在不同主机，以下列出两条记录，接下来定义了报告数据库的 DBID 和实例号：

```
Instances in this Workload Repository schema
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   DB Id     Inst Num DB Name      Instance     Host
------------ -------- ------------ ------------ ------------
* 1992983269        1 GHCCDB       ghccdb       ccdb1
* 1992983269        1 GHCCDB       ghccdb       ccdb2

Database Id and Instance Number for the First Pair of Snapshots
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Using 1992983269 for Database Id for the first pair of snapshots
Using          1 for Instance Number for the first pair of snapshots
```

接下来选择列出采样的时间，缺省列出全部：

```
Specify the number of days of snapshots to choose from
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Entering the number of days (n) will result in the most recent
(n) days of snapshots being listed.  Pressing <return> without
specifying a number lists all completed snapshots.

Enter value for num_days:

Listing all Completed Snapshots
                                                       Snap
Instance     DB Name        Snap Id    Snap Started    Level
------------ ------------ --------- ------------------ -----
ghccdb       GHCCDB            5275 13 Jul 2008 12:00      1
                               5276 13 Jul 2008 13:00      1
                               5277 13 Jul 2008 14:00      1
                               5278 13 Jul 2008 15:00      1
```

注意接下来提示与以往的不同，这里提示定义第一对起始和结束的快照 ID，这里选择问题时段的 5276～5277 时段：

```
Specify the First Pair of Begin and End Snapshot Ids
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Enter value for begin_snap: 5276
First Begin Snapshot Id specified: 5276

Enter value for end_snap: 5277
First End   Snapshot Id specified: 5277
```

接下来是和之前类似的过程，再次列出实例信息：

```
Instances in this Workload Repository schema
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   DB Id     Inst Num DB Name      Instance     Host
------------ -------- ------------ ------------ ------------
* 1992983269        1 GHCCDB       ghccdb       ccdb1
* 1992983269        1 GHCCDB       ghccdb       ccdb2
Database Id and Instance Number for the First Pair of Snapshots
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Using 1992983269 for Database Id for the first pair of snapshots
Using          1 for Instance Number for the first pair of snapshots
```

接下来同样列举采样数据：

```
Specify the number of days of snapshots to choose from
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
Entering the number of days (n) will result in the most recent
(n) days of snapshots being listed.  Pressing <return> without
specifying a number lists all completed snapshots.

Enter value for num_days2:
Listing all Completed Snapshots
                                                            Snap
Instance     DB Name        Snap Id    Snap Started    Level
------------ ------------ --------- ------------------ -----
ghccdb       GHCCDB            5275 13 Jul 2008 12:00      1
                               5276 13 Jul 2008 13:00      1
                               5277 13 Jul 2008 14:00      1
                               5278 13 Jul 2008 15:00      1
```

这里定义与之前不同的采样时段，选择 5277～5278 时段：

```
Specify the Second Pair of Begin and End Snapshot Ids
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Enter value for begin_snap2: 5277
Second Begin Snapshot Id specified: 5277

Enter value for end_snap2: 5278
Second End   Snapshot Id specified: 5278
```

最后定义输出报告名称，缺省的以 awrdiff 开头，也就是 AWR 报告对比之意：

```
Specify the Report Name
~~~~~~~~~~~~~~~~~~~~~~~
The default report file name is awrdiff_1_5276_1_5277.html  To use this name,
press <return> to continue, otherwise enter an alternative.

Enter value for report_name:

Using the report name awrdiff_1_5276_1_5277.html
```

现在看看这个生成的报告与普通报告的不同，首先第一部分题目显示“WORKLOAD REPOSITORY COMPARE PERIOD REPORT”，表示这是一个不同阶段的比较报告，第一个报告以及第二个报告的相关信息会对比列出，便于比较，如图 9-9 所示。

WORKLOAD REPOSITORY COMPARE PERIOD REPORT

Snapshot Set	DB Name	DB Id	Instance	Inst num	Release	Cluster	Host
First (1st)	GHCCDB	1992983269	ghccdb	1	10.2.0.3.0	NO	ccdb2
Second (2nd)	GHCCDB	1992983269	ghccdb	1	10.2.0.3.0	NO	ccdb2

Snapshot Set	Begin Snap Id	Begin Snap Time	End Snap Id	End Snap Time	Elapsed Time (min)	DB Time (min)	Avg Active Users
1st	5276	13-Jul-08 13:00:36	5277	13-Jul-08 14:00:48	60.20	43.80	0.73
2nd	5277	13-Jul-08 14:00:48	5278	13-Jul-08 15:00:01	59.22	885.31	14.71

Configuration Comparison

	1st	2nd	%Diff
Buffer Cache:	960M	960M	0.00
Std Block Size:	8K	8K	0.00
Shared Pool Size:	1,488M	1,488M	0.00
Log Buffer:	14,392K	14,392K	0.00
SGA Target:	2,617M	2,617M	0.00
PGA Aggregate Target:	1,063M	1,063M	0.00
Undo Management:	AUTO	AUTO	

图 9-9　AWR 对比报告

负载概要信息部分通过对比各类统计数据，可以直观地告诉我们不同时段数据库性能的变化，这是一个真实的诊断案例，注意逻辑读（Logical Reads）部分，第二个时段比第一个时段多出 194.24%，是第一个时段的近 3 倍，如果不是业务量的正常增长，那么就极有可能是系统出现异常，如图 9-10 所示。

Load Profile

	1st Per Sec	2nd Per Sec	%Diff	1st Per Txn	2nd Per Txn	%Diff
Redo size:	5,585.01	6,312.93	13.03	3,073.88	3,256.51	5.94
Logical reads:	28,871.28	84,949.93	194.24	15,890.21	43,821.22	175.77
Block changes:	26.40	27.35	3.60	14.53	14.53	0.00
Physical reads:	15.62	63.96	309.48	8.60	32.99	283.60
Physical writes:	6.48	51.24	690.74	3.57	26.43	640.34
User calls:	588.18	746.90	26.98	323.72	385.29	19.02
Parses:	176.00	217.97	23.85	96.87	112.44	16.07
Hard parses:	3.98	5.05	26.88	2.19	2.60	18.72
Sorts:	12.48	23.19	85.82	6.87	11.96	74.09
Logons:	0.03	0.12	300.00	0.02	0.06	200.00
Executes:	179.01	221.14	23.53	98.52	114.07	15.78
Transactions:	1.82	1.94	6.59			

	1st	2nd	Diff
% Blocks changed per Read:	0.09	0.03	-0.06
Recursive Call %:	27.08	25.41	-1.67
Rollback per transaction %:	0.05	0.90	0.85
Rows per Sort:	143.46	118.77	-24.69
Avg Length of Calls (sec):	0.00	0.00	0.00

图 9-10　负载概要信息

在这个案例中，继续检查 Buffer Gets Top 10 SQL，注意前两条 SQL，如图 9-11 所示，第一条 SQL 在第一个时段执行了 37 次，但是在第二个时段却执行了 3168 次；第二条 SQL 在第一个时段执行了 27 次，在第二个时段执行了 2928 次。这两个 Buffer Gets 在 2 万左右的 SQL 执行数量的激增导致了系统负荷急剧攀升。

Top 10 SQL Comparison by Buffer Gets

- Ordered by Diff column of % Total Gets descending.
- 'N/A' in this section indicates no data was captured for the statement in the period
- 'Multiple Plans' column in this section indicates whether more than one plan exist for the statement in the two periods
- Total Buffer Gets First:104287436, Second: 301840535

	% Total Gets			Gets / Exec		#Executions		Exec Time (ms) / Exec		CPU Time (ms) / Exec		Physical Reads / Exec		
SQL Id	1st	2nd	Diff	1st	2nd	1st	2nd	1st	2nd	1st	2nd	1st	2nd	SQL Text
g7c1vq4p34kty	0.79	23.38	22.59	22,349.05	22,275.50	37	3,168	297	5,627	297	301	0.00	0.00	select count(*) as x0_0_ from ...
fk3app5d0kwfn	0.61	22.92	22.31	23,608.04	23,646.50	27	2,926	363	7,355	361	367	0.00	0.00	select * from (select row_.*,...
6gm5gbhfqzk1a		1.89	1.89		23,558.25		242		1,961		370		0.00	select * from (select abstrac...
4su520du4ayf0	2.29	0.44	-1.85	22,142.13	22,349.15	108	60	299	634	296	305	0.00	0.00	select count(*) as x0_0_ from ...
ck9p2q2h1b1un	2.45	0.72	-1.73	29,710.60	28,390.90	86	77	822	940	809	790	0.00	0.00	select * from (select abstrac...
5tvra5q24awq9	2.06	0.39	-1.67	23,344.20	23,664.38	92	50	374	521	364	371	0.00	0.00	select * from (select row_.*,...
18j6k2cm89s7c	1.71	0.55	-1.17	20,789.84	21,478.36	86	77	308	363	306	321	0.00	0.00	select count(*) as x0_0_ from ...
aun3spdjpdqt3	0.81		-0.81	20,220.60		42		296		295		0.00		select count(*) as x0_0_ from ...

图 9-11　检查 Buffer Gets Top 10 SQL

回过头来看 Top 5 Time Events，如图 9-12 所示，在问题时段的 Latch 竞争极高，其中 latch: cache buffers chains 正是由于过量的 Buffer 扫描导致的，综合考虑，前面两个 SQL 的频繁执行正是性能问题的罪魁祸首，剩下的工作就很简单了，找到两个 SQL 频繁执行的原因，消除应用异常，系统即可恢复正常。

Top 5 Timed Events

1st					2nd				
Event	Waits	Time(s)	Percent Total DB Time	Wait Class	Event	Waits	Time(s)	Percent Total DB Time	Wait Class
CPU time		2,483.0	94.48		CPU time		5,883.0	11.08	
db file sequential read	26,033	69.5	2.64	User I/O	*latch: cache buffers chains	8,127	3,246.2	6.11	Concurrency
*SQL*Net more data from client	81,290	8.6	.33	Network	*latch free	1,587	612.0	1.15	Other
*cursor: pin S wait on X	538	8.6	.33	Concurrency	db file sequential read	31,685	80.9	.15	User I/O
*log file sync	6,385	5.0	.19	Commit	*direct path read temp	5,406	54.4	.10	User I/O
-latch free	26	0.2	.01	Other	-cursor: pin S wait on X	2,020	45.2	.09	Concurrency
-latch: cache buffers chains	248	0.1	.00	Concurrency	-SQL*Net more data from client	99,571	18.2	.03	Network
-direct path read temp	305	0.1	.00	User I/O	-log file sync	6,650	11.1	.02	Commit

图 9-12　查看 Top 5 Time Events

创建比较报告也可以通过 Database Control 来进行，在“主页-性能-其他监视链接”选择“快照”选项即可进入 AWR 数据页，在该页面选择“比较时段”后，即可开始创建不同时段的采样比较报告，如图 9-13 所示。

图 9-13　定义快照

在完成两个阶段的起始与结束快照定义之后即可确认完成，进行报告创建，如图 9-14 所示。

图 9-14　确认比较时段

为了能够通过比较机制对数据库不同阶段的性能情况进行比较，可以为 AWR 创建基线（Base Line），创建的基线不会被清除，以后生成的采样数据或者优化后采样可以同保留的基线进行对比，以确定数据库的性能变化。创建 Base Line 可以通过 Database Control 进行，也可以通过命令完成，在内部都是通过 DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE 来完成 Base Line 的创建：

```
PROCEDURE CREATE_BASELINE
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --------
 START_SNAP_ID                  NUMBER                  IN
 END_SNAP_ID                    NUMBER                  IN
 BASELINE_NAME                  VARCHAR2                IN
 DBID                           NUMBER                  IN     DEFAULT
```

创建的 Base Line 可以通过数据字典视图 dba_hist_baseline 查询。类似以前的 Statspack，Oracle 允许将 AWR 数据导出并迁移到其他数据库以便于以后分析。Oracle 10gR2 提供了新工具来完成导出和迁移 AWR 数据的工作。

DBMS_SWRF_INTERNAL.AWR_EXTRACT 可以用来导出数据，awrextr.sql 脚本就是用来完成这个工作的，而导入工作可以通过 DBMS_SWRF_INTERNAL 包中的 AWR_LOAD 和 MOVE_TO_AWR 过程来完成，awrload.sql 脚本用于完成这个工作。

5．RAC 环境 AWR 信息的对比展现

在 Oracle Database 11g 中，Oracle 引入了一个新的脚本工具 spawrrac.sql 用于收集 RAC 环境下的数据库对比信息，在某些情况下可以清晰的展现 RAC 环境中的一些问题，这个脚本同样可以用于 Oracle Database 10g，在 10.2.0.4 中使用一切正常，其他版本请测试后使用。

该脚本的文件说明信息如下(spawrrac 意即 Server Performance AWR RAC report)：

```
Rem $Header: spawrrac.sql 23-apr-2007.11:13:39 cgervasi Exp $
Rem
Rem spawrrac.sql
Rem
Rem Copyright (c) 2007, Oracle. All rights reserved.
Rem
Rem    NAME
Rem      spawrrac.sql - Server Performance AWR RAC report
Rem
Rem    DESCRIPTION
Rem      This scripts generates a global AWR report to report
Rem      performance statistics on all nodes of a cluster.
Rem
Rem    NOTES
Rem      Usually run as SYSDBA
```

运行和使用 awrrpt.sql 脚本类似：

```
SQL> @?/rdbms/admin/spawrrac.sql

Instances in this AWR schema
~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                         Instance
   DB Id    DB Name         Count
----------- ----------- --------
 4266683088 SMSDB               2

Enter value for dbid: 4266683088
```

```
Using 4266683088 for database Id

Specify the number of days of snapshots to choose from
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Entering the number of days (n) will result in the most recent
(n) days of snapshots being listed.  Pressing <return> without
specifying a number lists all completed snapshots.

Listing the last 31 days of Completed Snapshots

                                          Snap Instance
DB Name        Snap Id End Interval Time Level    Count
------------ --------- ---------------- ----- --------
.....
SMSDB             9566 02 Feb 2009 00:00     1        2
                  9567 02 Feb 2009 01:00     1        2
                  9568 02 Feb 2009 02:00     1        2
                  9569 02 Feb 2009 03:00     1        2
                  9570 02 Feb 2009 04:00     1        2
                  9571 02 Feb 2009 05:00     1        2
                  9572 02 Feb 2009 06:00     1        2
                  9573 02 Feb 2009 07:00     1        2


Specify the Begin and End Snapshot Ids
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Enter value for begin_snap: 9572
Begin Snapshot Id specified: 9572

Enter value for end_snap: 9573
End   Snapshot Id specified: 9573

Specify the Report Name
~~~~~~~~~~~~~~~~~~~~~~~
The default report file name is spawrrac_9572_9573.  To use this name,
press <return> to continue, otherwise enter an alternative.

Enter value for report_name:
```

```
Using the report name spawrrac_9572_9573
```

生成的报告中，对于 OS 系统信息以及 RAC 信息具有清晰的对比展现，可以很容易发现 RAC 环境中的异常及性能问题，以下对前面生成的报告输出做简要说明。

首先的 OS 统计信息中我们就可以发现，两个节点主机的繁忙程度严重不同，实例 2 非常繁忙，这说明两个节点负载不均衡，也可能节点 2 上有定时的任务执行：

```
OS Stat
~~~~~~~
          Num    Load    Load
Inst #  CPUs   Begin     End  % Busy  % Usr  % Sys  % WIO  % Idl  Busy Time (s)  Idle Time (s)  Total time (s)
------  ----  ------  ------  ------ ------ ------ ------ ------ --------------- -------------- ---------------
     1     4      .3      .5     2.6    1.1    1.5     .0   97.4          378.61      13,974.33       14,352.94
     2     4     1.2     1.2    26.9   10.2   16.7     .0   73.1        3,861.20      10,492.55       14,353.75
                                                                 --------------- -------------- ---------------
sum                                                                     4,239.81      24,466.88       28,706.69
```

进一步的在 Global Cache 的信息中，实例 2 请求了大量的跨实例访问的数据块，这进一步说明了实例 2 上存在大规模的查询或任务操作：

```
SysStat and GE Misc - RAC
~~~~~~~~~~~~~~~~~~~~~~~~~

    GC Current      GC CR  GC Current      GC CR
        Blocks     Blocks      Blocks     Blocks                    IPC  GC Messages GE Messages   Msgs Rcvd
I#    Received   Received      Served     Served GC CPU (s)   CPU (s)         Sent        Sent      Actual
---- ---------- ---------- ---------- ---------- ---------- ---------- ------------ ----------- -----------
   1      3,322      1,736  2,089,116        608        208          0       27,744       4,621     309,873
   2  2,091,838        612      3,603      1,764        168          0    2,767,283       4,629      28,000
     ---------- ---------- ---------- ---------- ---------- ---------- ------------ ----------- -----------
sum   2,095,160      2,348  2,092,719      2,372        375          0    2,795,027       9,250     337,873
```

在后面的 SQL 展现中，我们发现如下一条 SQL 是导致大量 CPU 使用以及逻辑读的 SQL，这是一个物化视图的刷新引起的：

```
SQL ordered by Elapsed Time (Global)
-> Total DB time (s):          1,658
-> Captured SQL accounts for    98.4% of Total DB time

SQL Id                Ela (s)      CPU (s)          Gets      Execs SQL Text
------------- ------------- ----------- -------------- ---------- ---------------------------------------------------
              per Exec(s) per Exec(s)       per Exec
------------- ------------- ----------- -------------- ---------- ---------------------------------------------------
               % of DB time  % of DB CPU     % of Gets % of Execs
------------- ------------- ----------- -------------- ---------- ---------------------------------------------------
06marcjmvpbwq      1,583.78       201.47     4,734,267         78 select rptstatusstr status, to_char(rptdate, 'yyyy
                      20.30         2.58      60,695.7            -mm-dd hh24:mi:ss') rptdate from t_mt_log where li
                      95.53        85.94         54.68       0.17 nkid = :1
```

6. AWR 使用信息报告

Oracle 还随软件提供一个脚本用于输出 AWR 的使用信息，这个脚本是 awrinfo.sql，运行这个脚本，将会输出 AWR 的空间使用、快照采样、ASH 及 ADDM 等 AWR 元数据信息。输出显示为 3 类：

- AWR Snapshot Info Gathering；
- Advisor Framework Diagnostics；
- AWR and ASH Usage Info Gathering。

下面是运行这个脚本的输出的摘要示例：

```
SQL> @?/rdbms/admin/awrinfo.sql
```

```
~~~~~~~~~~~~~~~
AWR INFO Report
~~~~~~~~~~~~~~~
       DB_ID DB_NAME   HOST_PLATFORM             INST STARTUP_TIME     LAST_ASH_SID PAR
------------ --------- ------------------------- ---- --------------- ------------ ---
* 2590148133 EYGLE      eygle - Linux IA (32-bit)  1 19:55:16 (06/05)      1585852 NO

########################################################
(I) AWR Snapshots Information
########################################################

*****************************************************
(1a) SYSAUX usage - Schema breakdown (dba_segments)
*****************************************************
|
| Total SYSAUX size           186.6 MB ( 1% of 32,769.0 MB MAX with AUTOEXTEND ON )
|
| Schema  SYS          occupies           112.4 MB (  60.3% )
| Schema  SYSMAN       occupies            59.6 MB (  31.4% )
| Schema  WMSYS        occupies             6.9 MB (   3.7% )
| Schema  SYSTEM       occupies             6.8 MB (   3.7% )
| Schema  DBSNMP       occupies             1.6 MB (   0.8% )
| Schema  TSMSYS       occupies             0.3 MB (   0.1% )
|
**********************************
(3b) Space usage within AWR Components (> 500K)
**********************************

COMPONENT        MB SEGMENT_NAME - % SPACE_USED                          SEGMENT_TYPE
--------- --------- --------------------------------------------------- ---------------
FIXED           2.0 WRH$_SYSMETRIC_SUMMARY                        -  85%  TABLE
FIXED           0.9 WRH$_SYSMETRIC_SUMMARY_INDEX                  -  90%  INDEX
FIXED           0.6 WRH$_LATCH.WRH$_LATCH_2590148133_266          -  83%  TABLE PARTITION
FIXED           0.6 WRH$_LATCH.WRH$_LATCH_2590148133_97           -  83%  TABLE PARTITION
FIXED           0.6 WRH$_LATCH.WRH$_LATCH_2590148133_121          -  83%  TABLE PARTITION
FIXED           0.6 WRH$_LATCH.WRH$_LATCH_2590148133_242          -  84%  TABLE PARTITION
FIXED           0.6 WRH$_LATCH.WRH$_LATCH_2590148133_145          -  83%  TABLE PARTITION
FIXED           0.6 WRH$_LATCH.WRH$_LATCH_2590148133_169          -  87%  TABLE PARTITION
FIXED           0.6 WRH$_LATCH.WRH$_LATCH_2590148133_194          -  83%  TABLE PARTITION
```

```
FIXED          0.6 WRH$_LATCH.WRH$_LATCH_2590148133_218      -  83%  TABLE PARTITION
SQL            0.6 WRH$_SQLSTAT.WRH$_SQLSTA_2590148133_169   -  84%  TABLE PARTITION
SQL            0.5 WRH$_SQLSTAT.WRH$_SQLSTA_2590148133_97    -  87%  TABLE PARTITION
SQL            0.5 WRH$_SQLSTAT.WRH$_SQLSTA_2590148133_266   -  85%  TABLE PARTITION
```

9.3.4 自动数据库诊断监控：ADDM 的引入

有了这个 AWR 这个“数据仓库”之后，Oracle 自然可以在此基础之上实现更高级别的智能应用，更大程度地发挥 AWR 的作用，这就是 Oracle 10g 引入的另外一个功能自动数据库诊断监控程序（Automatic Database Diagnostic Monitor，ADDM），通过 ADDM，Oracle 试图使数据库的维护、管理和优化工作变得更加自动和简单。

ADDM 可以定期检查数据库的状态，根据内建的专家系统，自动确定潜在的数据库性能瓶颈，并提供调整措施和建议。由于这一切都是内建在 Oracle 数据库系统之内的，其执行效率很高，几乎不影响数据库的总体性能。

新版的 Database Control 可以以一种方便直观的形式提供 ADDM 的结果和建议，并引导管理员逐步实施 ADDM 的建议，快速解决性能问题。

通过图 9-15 可以直观地看到 AWR 及 ADDM 的关系。

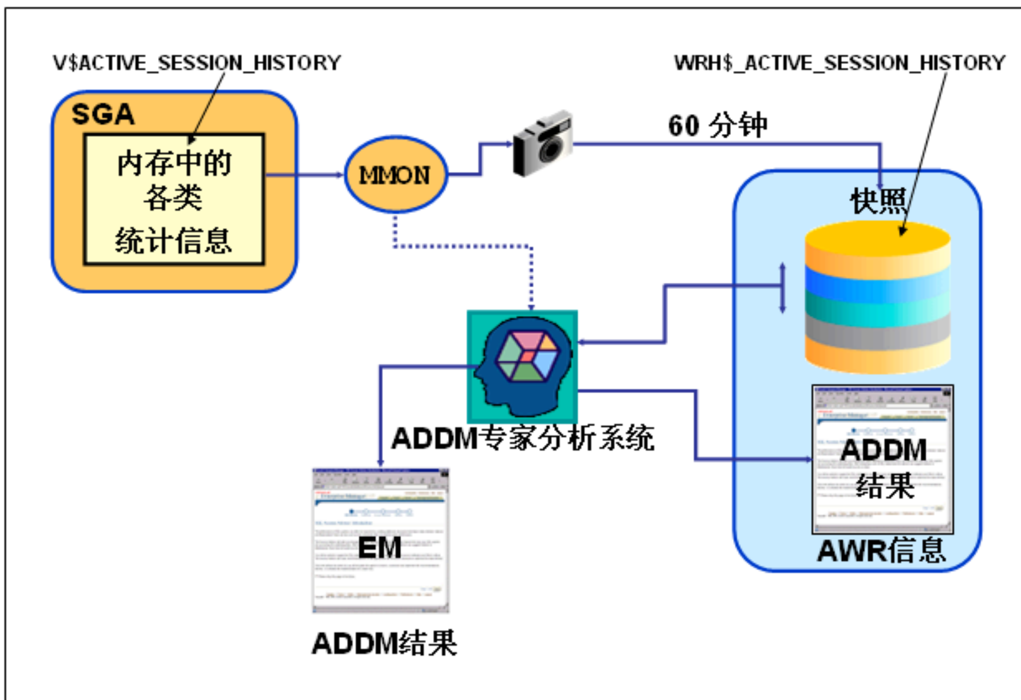


图 9-15　AWR 及 ADDM 的关系

对于 ADDM，本章不打算做过多的详细介绍。

9.4　顶级等待事件

前文还提到另外一个重要视图 V$SYSTEM_EVENT，该视图记录的是数据库自启动以来等待事件的汇总。通过查询该视图，就可以快速获得数据库等待事件的总体概况，了解数据库运行的基本状态：

```
SQL> SELECT *
  2    FROM (SELECT   event, time_waited
  3              FROM v$system_event
  4          ORDER BY time_waited DESC)
  5   WHERE ROWNUM < 11;
EVENT                                                              TIME_WAITED
---------------------------------------------------------------- -----------
SQL*Net message from client                                        9.2256E+10
rdbms ipc message                                                  1.2383E+10
pmon timer                                                         1834857492
smon timer                                                         1771582338
jobq slave wait                                                      414242315
db file scattered read                                                  54344796
enqueue                                                            29142826
latch free                                                           18022667
db file sequential read                                                 11925101
log file sync                                                          9500670

10 rows selected.
```

以上是一个产品环境中的 Top10 等待事件，我们注意到 Top5 等待都是空闲等待，所以不必过多关注，但是接下来的 5 个等待事件都是常见的重要等待事件，如果能够进行针对性优化，数据库性能将会得到大幅提升。

在 Oracle 的 Statspack Report 中，有一部份信息为 Top 5 Wait Events（在 Oracle 9i 中更改为 Top 5 Time Events），这部分信息就是来自 V$SYSTEM_EVENT 视图的采样。

以下是一个 Statspack 的诊断报告：

```
DB Name        DB Id      Instance    Inst Num  Release     OPS   Host
----------  -----------  ----------  --------  ----------  ----  ----------
K2         1999167370      k2              1  9.1.5.0.0   NO      k2
```

这是一个 9.1.5 的数据库系统，通过脚本增强，可以在 9.1.5 的数据库上使用 Statspack 来进行数据库诊断。

```
                                                                Snap Length
Start Id    End Id       Start Time             End Time         (Minutes)
--------  --------  --------------------  --------------------  -----------
     170       176  25-Feb-03 10:00:11    25-Feb-03 15:00:05         299.90
```

```
Cache Sizes
~~~~~~~~~~~
           db_block_buffers:       64000
              db_block_size:        8192
                 log_buffer:     8388608
           shared_pool_size:   157286400
……
Top 5 Wait Events
~~~~~~~~~~~~~~~~~                                      Wait     % Total
Event                                          Waits  Time (cs)   Wt Time
-------------------------------------------- ------------ ---------------------- -------
db file scattered read                         16,842,920    3,490,719   43.32
latch free                                     844,272    3,270,073   40.58
buffer busy waits                               114,421      933,136   11.58
db file sequential read                          2,067,910      117,750    1.46
enqueue                                          464      110,840    1.38
         -------------------------------------------------------------
```

这里的 Top 5 Wait Events 是诊断的重要依据。这是一个典型的性能低下的系统，几个重要的等待事件都在 Top 5 中出现，其中，前 3 个等待极为显著，需要进行相应的调整。

在 5 小时的采样间隔内，其中 db file scattered read 累计等待时间约 10 小时，已经成为影响系统性能的主要原因。了解了这些以后就可以进一步查看 Statspack Report 中相关 SQL 部分，看是否存在可疑的 SQL 语句。

```
SQL ordered by Gets for DB: K2  Instance: k2  Snaps:     170 -    176
                                Gets       % of
   Buffer Gets     Executes    per Exec    Total   Hash Value
-------------- ------------ ------------ ------ ------------
SQL statement
------------------------------------------------------------------------------
     6,480,163           12    540,013.6    2.4   3791855498
  SELECT "PROCESS_REQ"."WORK_ID", "PROCESS_REQ"."STOCK_NO",    "PROCESS_R
     3,784,566           16    236,535.4    1.4   2932917818
SELECT * FROM FIND_LATER_WO ORDER BY NOTE,ORDER_NO
     1,200,976            3    400,325.3     .4   4122791109
  SELECT "ITEM_STOCK"."ITEM_NO",              "ITEM"."NOTE",            "ITEM"
       923,944            9    102,660.4     .3   2200071737
  SELECT  "ITEM_STOCK"."ITEM_NO" ,            "ITEM_STOCK"."STOCK_NO" ,
       921,301            3    307,100.3     .3   2218843294
  SELECT "ITEM_STOCK"."ITEM_NO",              "ITEM"."NOTE",            "ITEM"
       911,285            3    303,761.7     .3   1769130587
```

```
  SELECT  "LISTS"."ITEM_NO" ,           "LISTS"."SUB_ITEM" ,           "LISTS"
       831,439            2    415,719.5     .3   1349577999
  SELECT  "GROUP_OPER"."ITEM_NO" ,           "GROUP_OPER"."PROCESS_ID" ,
       802,918            1    802,919.0     .3   3613809507
  SELECT  "LISTS"."ITEM_NO" ,           "LISTS"."SUB_ITEM" ,           "ITEM".
       800,548            2    400,274.0     .3   2643788247
  SELECT "ITEM_STOCK"."ITEM_NO",              "ITEM"."NOTE",            "ITEM"
       666,085            2    333,042.5     .2   3391363608
  SELECT "ITEM_STOCK"."ITEM_NO",              "ITEM_STOCK"."STOCK_NO",
          …….
```

注意到以上很多查询导致的 Buffer Gets 都非常庞大，我们非常有理由怀疑索引存在问题，甚至缺少必要的索引。以上记录的是 SQL 的片段，通过 Hash Value 值结合 v$sql_text 可以获得完整的 SQL 语句（可以参考前文的案例）。

在这次诊断中，我紧接着去查询的是 v$session_longops 视图，一个分组查询的结果如下：

```
TARGET                                  COUNT(*)
---------------------------------------------------------------- ----------
SA.PPBT_GRAPHOBJTABLE                  418
SA.PPBT_PPBTOBJRELATTABLE              53
```

发现这些问题 SQL 的全表扫描（结合 v$session_longops 视图中的 OPNAME）主要集中在 PPBT_GRAPHOBJTABLE 和 PPBT_PPBTOBJRELATTABLE 两张数据表上。进一步研究发现这两个数据表上没有任何索引，并且有相当的数据量：

```
SQL> select count(*) from SA.PPBT_PPBTOBJRELATTABLE;
  COUNT(*)
----------
   1209017
SQL> select count(*) from SA.PPBT_GRAPHOBJTABLE;
   COUNT(*)
----------
      2445
```

在创建了合适的索引后，系统性能得到了大幅提高!

9.5　重要等待事件

在了解了等待事件的作用和变迁之后，让我们来了解一下重要的等待事件。

9.5.1 db file sequential read-数据文件顺序读取

db file sequential read 是个非常常见的 I/O 相关的等待事件，通常显示与单个数据块相关的读取操作，在大多数的情况下，读取一个索引块或者通过索引读取一个数据块时，都会记录这个等待。

这个等待事件有 3 个参数 P1、P2、P3，其中 P1 代表 Oracle 要读取的文件的绝对文件号，P2 代表 Oracle 从这个文件中开始读取的起始数据块块号，P3 代表读取的 BLOCK 数量，通常这个值为 1，表明是单个 BLOCK 被读取。

```
SQL> select name,parameter1,parameter2,parameter3
  2  from v$event_name where name='db file sequential read';
NAME            PARAMETER1 PARAMETER2 PARAMETER3
------------------------------ ---------- ---------- ----------
db file sequential read        file#      block#     blocks
```

在 Oracle 10g 中，这个等待事件被归入 User I/O 一类：

```
SQL> select name,wait_class
  2  from v$event_name where name='db file sequential read';
NAME                   WAIT_CLASS
------------------------------ -------------------------
db file sequential read        User I/O
```

图 9-16 简要说明了单块读取的操作方式。

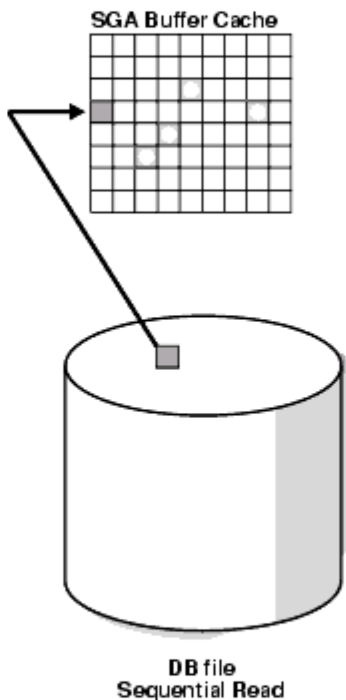


图 9-16　单块读取的操作

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，所以对于一个编码规范、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。

从 Oracle 9iR2 开始，Oracle 引入了段级统计信息收集的新特性，可以通过查询 V$SEGMENT_STATISTICS 视图，找出物理读取显著的索引段或者是表段，研究其数据结构，看能否通过重建或者重新规划分区、存储参数等手段降低其 I/O 访问。

Oracle 9iR2 中，收集的统计信息共有 11 类：

```
SQL> select * from v$segstat_name;
STATISTIC# NAME                                SAMPLED
---------- -------------------------------------- ----------
         0 logical reads                          YES
         1 buffer busy waits                      NO
         2 db block changes                       YES
         3 physical reads                         NO
         4 physical writes                        NO
         5 physical reads direct                  NO
         6 physical writes direct                 NO
         8 global cache cr blocks served          NO
         9 global cache current blocks served     NO
        10 ITL waits                              NO
        11 row lock waits                         NO
11 rows selected.
```

在 Oracle 10gR2 中，这类统计信息增加为 15 个：

```
SQL> select * from v$segstat_name;
STATISTIC# NAME                           SAM
---------- ------------------------------ ---
         0 logical reads                  YES
         1 buffer busy waits              NO
         2 gc buffer busy                 NO
         3 db block changes               YES
         4 physical reads                 NO
         5 physical writes                NO
         6 physical reads direct          NO
         7 physical writes direct         NO
         9 gc cr blocks received          NO
```

```
        10 gc current blocks received      NO
        11 ITL waits                       NO
        12 row lock waits                  NO
        14 space used                      NO
        15 space allocated                 NO
        17 segment scans                   NO
15 rows selected.
```

对于 CBO 模式下的数据库，应当及时收集统计信息，使 SQL 可以选择正确的执行计划，避免因为统计信息陈旧而导致的执行错误等。

9.5.2 db file scattered read 等待事件

在前面的案例中，已经多次见到 db file scattered read 等待事件，在生产环境之中，这个等待事件可能更为常见。这个事件表明用户进程正在读数据到 Buffer Cache 中，等待直到物理 I/O 调用返回。DB File Scattered Read 发出离散读，将存储上连续的数据块离散的读入到多个不连续的内存位置。Scattered Read 通常是多块读，在 Full Table Scan 或 Fast Full Scan 等访问方式下使用。

Scattered Read 代表 Full Scan，当执行 Full Scan 读取数据到 Buffer Cache 时，通常连续的数据在内存中的存储位置并不连续，所以这个等待被命名为 Scattered Read（离散读）。每次多块读读取的数据块数量受初始化参数 DB_FILE_MULTIBLOCK_READ_COUNT 限制。图 9-17 简要说明了 Scattered Read 的数据读取方式。

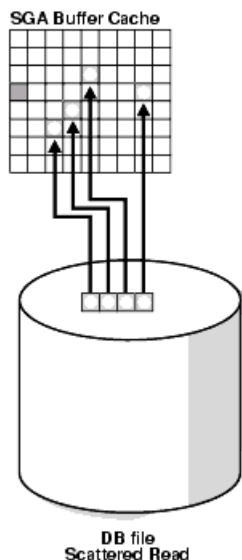


图 9-17　Scattered Read 的数据读取

从 V$EVENT_NAME 视图可以看到，该等待有 3 个参数，分别代表文件号、起始数据块号、数据块的数量：

```
SQL> select * from v$event_name where name='db file scattered read';
EVENT#       NAME                   PARAMETER1             PARAMETER2       PARAMETER3
----------   --------------------- ---------------------  -------------    ----
206          db file scattered read    file#                  block#         blocks
```

数据文件号、起始数据块号加上数据块的数量，通过这些信息可以知道 Oracle Session 正在等待的对象文件等信息。该等待可能和全表扫描（Full Table Scan）或者快速全索引扫描（Index Fast Full Scan）的连续读取相关，根据经验，通常大量的 db file scattered read 等待可能意味着应用问题或者索引缺失。

在实际环境的诊断过程中，可以通过 v$session_wait 视图发现 Session 的等待，再结合其他视图找到存在问题的 SQL 等根本原因，从而从根本上解决问题。此类诊断案例，可以参考 9.2 小节的内容。

当这个等待事件比较显著时，用户也可以结合 v$session_longops 动态性能视图来进行诊断，该视图中记录了长时间（运行时间超过 6 秒的）运行的事务，可能很多是全表扫描操作（不管怎样，这部分信息都是值得我们注意的），上一个案例就是通过 v$session_longops 快速发现了问题所在。

从 Oracle 9i 开始，Oracle 新增加了一个视图 V$SQL_PLAN 用于记录当前系统 Library Cache 中 SQL 语句的执行计划，可以通过这个视图找到存在问题的 SQL 语句，以下是在一个生产系统中查询得到的结果：

```
SQL> @getplan
Enter value for waitevent: free buffer waits
old  15:                           AND b.event = '&waitevent')
new  15:                           AND b.event = 'free buffer waits')

HASH_VALUE CHILD_NUMBER OPERATION        OBJECT              COST    KBYTES
---------     ------------ ---------------------------------------- -------------------
2838180055    0 INSERT STATEMENT CHOOSE Cost=41733                            41733
2838180055    0   TABLE ACCESS FULL           I_CM_POWER_TEMP      41733   1356468
```

进而可以通过 v$sql_text 视图获得这个问题 Session 正在执行的 SQL 语句：

```
SQL> select sid,event from v$session_wait;
       SID EVENT
---------- ----------------------------------------------------------------
         1 pmon timer
         4 rdbms ipc message
         7 rdbms ipc message
         5 rdbms ipc message
         8 rdbms ipc message
        21 free buffer waits
        49 free buffer waits
         2 db file parallel write
```

```
         3 db file parallel write
         6 smon timer
    ……
16 rows selected.
SQL>@ GetSqlBySid
Enter value for sid: 49
old   5: where b.sid='&sid'
new   5: where b.sid='49'
SQL_TEXT
------------------------------------------------------------------------
insert into i_cm_power_new(PNAME,YYS,SPHM,SJH,SENTTIME,NOTES,PLACE,RMK)
select PNAME,YYS,SPHM,SJH,SENTTIME,NOTES,PLACE,RMK FROM i_cm_power_temp
```

通过 V$SQL_PLAN 视图，可以获得大量有用的信息，比如获得全表扫描的对象：

```
SQL> select distinct object_name,object_owner from v$sql_plan p
  2  where p.operation='TABLE ACCESS' and p.options='FULL'
  3  and object_owner = 'MKT';
OBJECT_NAME                                         OBJECT_OWNER
--------------------------------------------------- -----------------
HD_TEMP                                                     MKT
I_CM_BILL                                                   MKT
I_CM_IVR_BUTTON                                             MKT
……
TOOLS_HD                                                    MKT
TOOLS_HD_NEW                                                MKT
TOOLS_HD_NEW_BAK                                            MKT
TOOLS_IVRBLIST                                              MKT
TOOLS_USER_CANCEL                                           MKT
29 rows selected
```

或者获得全索引扫描对象：

```
SQL> select distinct object_name,object_owner from v$sql_plan p
  2  where p.operation='INDEX' and p.options='FULL SCAN'   ;
OBJECT_NAME                    OBJECT_OWNER
------------------------------ -----------------------------------
FK_ITEM_LEVEL_CODE             AVATAR
FK_ITEM_SELLCNT_CODE           AVATAR
FK_MYZZIM_CRTDATE              AVATAR
I_SYSAUTH1                     SYS
SYS_C008211                    WLLM
```

进而可以通过 V$SQL_PLAN 和 V$SQLTEXT 联合，获得这些查询的 SQL 语句，查找全

表扫描的 SQL 语句可以参考如下语句：

```
SELECT   sql_text  FROM v$sqltext t, v$sql_plan p
   WHERE t.hash_value = p.hash_value  AND p.operation = 'TABLE ACCESS' AND p.options = 'FULL'
ORDER BY p.hash_value, t.piece;
```

查找 Fast Full Index 扫描的 SQL 语句可以参考如下语句：

```
SELECT   sql_text FROM v$sqltext t, v$sql_plan p
   WHERE t.hash_value = p.hash_value AND p.operation = 'INDEX' AND p.options = 'FULL SCAN'
ORDER BY p.hash_value, t.piece;
```

这些信息对于发现数据库问题，优化数据库性能具有极强的指导意义。本例中用到的 SQL 代码 getplan.sql 内容如下：:

```
SET linesize 120
COL operation     format a55
COL cost          format 99999
COL kbytes        format 999999
COL object        format a25
SELECT   hash_value, child_number, LPAD (' ', 2 * DEPTH) || operation || ' ' || options
         || DECODE (ID, 0, SUBSTR (optimizer, 1, 6) || ' Cost=' || TO_CHAR (COST) ) operation,
         object_name OBJECT, COST, ROUND (BYTES / 1024) kbytes
    FROM v$sql_plan WHERE hash_value IN (
                    SELECT a.sql_hash_value FROM v$session a, v$session_wait b
                     WHERE a.SID = b.SID  AND b.event = '&waitevent')
ORDER BY hash_value, child_number, ID;
```

在 Oracle 10g 中，Oracle 对等待事件进行了分类，db file scattered read 事件被归入 User I/O 一类：

```
SQL> select name,PARAMETER1 p1,PARAMETER2 p2,PARAMETER3 p3,
  2  WAIT_CLASS_ID,WAIT_CLASS#,WAIT_CLASS
  3  from v$event_name where name='db file scattered read';
NAME                       P1       P2       P3       WAIT_CLASS_ID WAIT_CLASS# WAIT_CLASS
------------------------- -------- -------- -------- ------------- ----------- ----------
db file scattered read    file#    block#   blocks      1740759767           8 User I/O
```

完成对等待事件的分类之后，Oracle 10g 的 ADDM 可以很容易地通过故障树分析定位到问题所在，帮助用户快速发现数据库的瓶颈及瓶颈的根源，这就是 Oracle 的 ADDM 专家系统的设计思想。

通过图 9-18 可以直观而清晰地看到这个等待模型和 ADDM 结合实现的 Oracle 专家诊断系统。

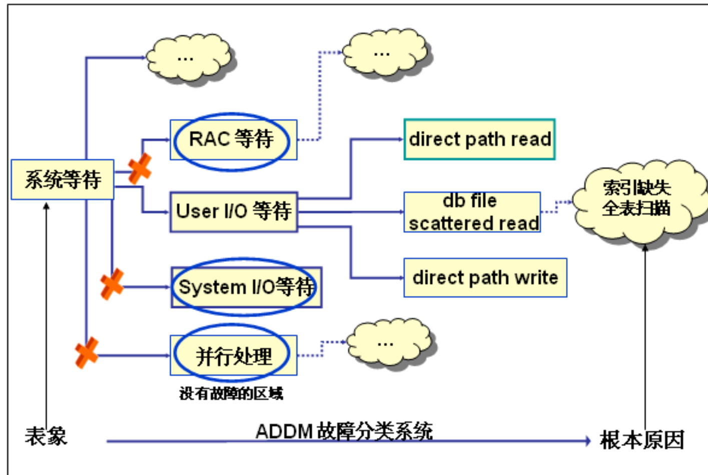


图 9-18　Oracle 专家诊断系统

9.5.3 direct path read /write（直接路径读/写）

直接路径读（direct path read）通常发生在 Oracle 直接读数据到进程 PGA 时，这个读取不需要经过 SGA。直接路径读等待事件的 3 个参数分别是 file#（指绝对文件号）、first block#、block 数量。在 Oracle 10g 中，这个等待事件被归于 User I/O 一类。

db file sequential read、db file scattered read、direct path read 是常见的集中数据读方式，图 9-19 简要描述了这 3 种方式的读取示意。

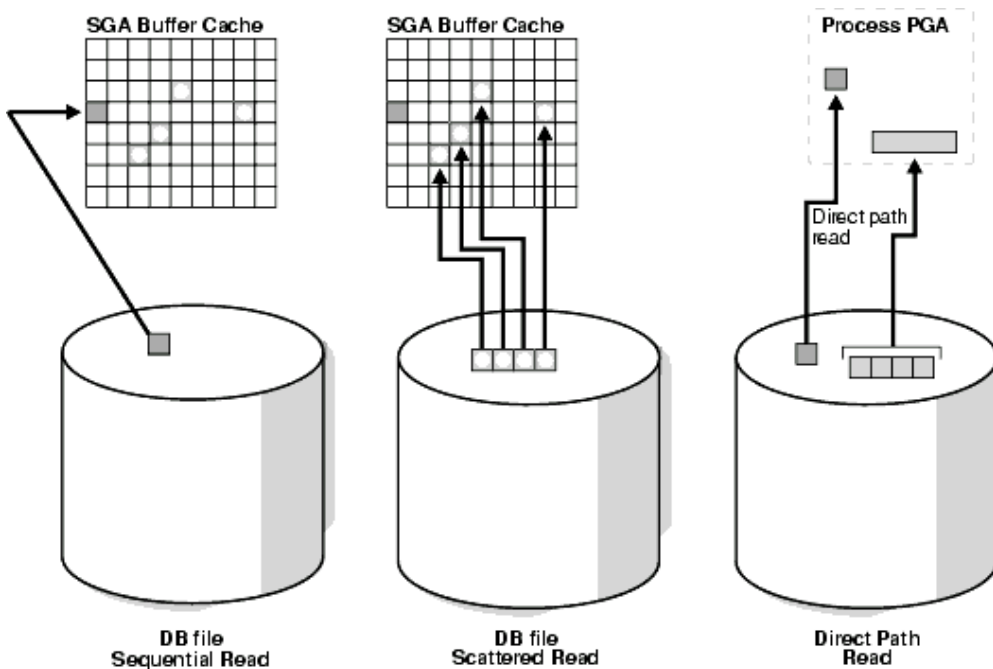


图 9-19　3 种读取方式

这类读取通常在以下情况被使用：

- 磁盘排序 IO 操作；
- 并行查询从属进程；
- 预读操作；
- 串行全表扫描（Oracle 11g 新特性）

最为常见的是第一种情况。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 path 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。直接路径写等待事件的 3 个参数分别是 file#（指绝对文件号）、first block#和 block 数量，在 Oracle 10g 中，这个等待事件同 direct path read 一样被归于 User I/O 一类。

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。

1．磁盘排序诊断案例

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，可以考虑为不同用户分配不同的临时表空间，使用多个临时文件，写入不同磁盘或者裸设备，从而降低竞争，提高性能；对于 Oracle 8i 的数据库，应该考虑使用本地管理（Local）的临时表空间，而不是字典（DICTIONARY）管理。

从 dba_tablespaces 视图可以获得这部分信息：

```
SQL> select tablespace_name,EXTENT_MANAGEMENT from dba_tablespaces;
TABLESPACE_NAME              EXTENT_MAN
------------------------------ ----------
SYSTEM                         DICTIONARY
RBS                            DICTIONARY
TEMP                           DICTIONARY
USERS                          LOCAL
4 rows selected.
```

可以看一个 Statspack 报告的典型例子：

```
DB Name        DB Id      Instance    Inst Num  Release     OPS   Host
----------  -----------  ----------  --------  ----------  ----  ----------
DB       294605295       db            1   9.1.5.0.0   NO      IBM
```

```
                                                                Snap Length
Start Id    End Id       Start Time             End Time         (Minutes)
--------  --------  --------------------  --------------------  -----------
      65        66  08-11月-03 16:32:42   08-11月-03 16:54:00         21.30
```

这是一个 20 分钟的采样报告，我们看到 direct path read/write 的等待都很显著：

```
Top 5 Wait Events
~~~~~~~~~~~~~~~~~                                         Wait     % Total
Event                                               Waits  Time (cs)   Wt Time
-------------------------------------------- ------------ ------------ -------
direct path write                                  98,631         3,651   44.44
log file switch completion                             62         2,983   36.31
direct path read                                   37,434         1,413   17.20
db file sequential read                                86           109    1.33
control file sequential read                        3,862            34     .41
         -------------------------------------------------------------
```

这可能意味着系统存在着大量的磁盘排序操作。基于此，继续向下追查相关排序部分统计数据：

```
Instance Activity Stats for DB:  DB  Instance:  db  Snaps:      65 -
Statistic                                      Total   per Second    per Trans
--------------------------------- ---------------- ------------ ------------
…….
sorts (disk)                                      64          0.1          0.4
sorts (memory)                                   861          0.7          4.7
sorts (rows)                               2,804,580      2,194.5     15,159.9
```

64 次的 sort disk，相当显著的磁盘排序。

在 Statspack 的报告中，存在一个性能指标，称为内存排序率（In-memory Sort Ratio），用于衡量系统的排序操作，这个指标就是由以上两个统计信息 Sort (disk)和 Sort（memory）得出：

In-memory Sort Ratio = Sort（memory）/ [sorts（disk） + Sort（memory）]

对于本例，这个比率计算值为：

In-memory Sort Ratio = 861 / （861 + 64）≈ 93.08 %

从 Statspack 的报告中，也可以获得这个信息：

```
Instance Efficiency Percentages (Target 100%)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        Buffer Nowait Ratio:      100.00
        Buffer  Hit   Ratio:       20.27
        Library Hit   Ratio:       99.81
        Redo  NoWait Ratio:      99.74
        In-memory Sort Ratio:      93.08
           Soft Parse Ratio:       99.66
```

```
           Latch Hit Ratio:      100.00
```

对于显著的磁盘排序，可以很容易地猜测到，临时表空间的读写操作肯定相当频繁，从 Statspack 报告中文件 I/O 部分的统计数据可以验证：

```
File IO Statistics for DB: GHCXSDB  Instance: ghcxsdb  Snaps:      65 -     66
Tablespace               Filename
------------------------ ----------------------------------------------------
         Reads Avg Blks Rd Avg Rd (ms)         Writes  Tot Waits Avg Wait (ms)
-------------- ----------- ----------- -------------- ---------- -------------
…….
PERFSTAT                 D:\ORACLE\ORADATA\PERFSTAT.DBF
            88         1.0        12.5            821          0
RBS                      D:\ORACLE\ORADATA\GHCXSDB\RBS01.DBF
             7         1.0         0.0          1,399          0
SYSTEM                   D:\ORACLE\ORADATA\GHCXSDB\SYSTEM01.DBF
            17         1.0        11.8             50          0
TEMP                     D:\ORACLE\ORADATA\GHCXSDB\TEMP01.DBF
       223,152         1.5         0.2        371,303          0
```

对于这种情况，在 Oracle 9i 之前，可以适当增加 sort_area_size 的大小；从 Oracle 9i 开始，可以适当增大 pga_aggregate_target，以缩减磁盘排序对于硬盘的写入，从而提高系统及应用相应。但是通常应该及时检查应用，确认是否因为应用问题而导致了过度排序，从而从根本上解决问题。

2．并行查询导致性能问题一则

有时候在应用系统中，不正确的使用并行查询也会导致应用问题，以下是一个实际生产中的案例。Statspack 的 Top 5 时间事件输出显示 direct path read 消耗了较高的等待：

```
Top 5 Timed Events
~~~~~~~~~~~~~~~~~~                                                  % Total
Event                                               Waits    Time (s) Ela Time
-------------------------------------------- ------------ ----------- --------
CPU time                                                        5,035    39.49
db file sequential read                           444,011       3,582    29.09
direct path read                                  124,451       1,351    10.60
db file scattered read                            389,933         908     7.12
KJC: Wait for msg sends to complete                31,985         600     4.70
          -------------------------------------------------------------
```

而这个数据库的内存排序率是 100%（In-memory Sort %: 100.00），显然这里的 Direct Path Read 并不是由于排序引发的，注意到另外一个等待事件（KJC: Wait for msg sends to complete）和并行有关，所以初步判断这里的 direct path read 可能和并行有关。

进一步检查 Statspack 报告中的 SQL 部分，发现大量并行查询改写出来的 SQL，这些 SQL

通过内部提示（Hints）固化其执行路径：

```
        250,458          133        1,883.1   11.9     3.30     29.80  883303536
Module: yy_glxt.exe
SELECT /*+ ORDERED NO_EXPAND USE_NL(A2) INDEX(A2 "SYS_C005617")
*/ A1.C0,A1.C1,A1.C2,A1.C3,A1.C4,A1.C5,A1.C6,A1.C7,A1.C8,A1.C9,A
1.C10,A1.C11,A1.C12,A1.C13,A1.C14,A1.C15,A1.C16,A1.C17,A1.C18,A1
.C19,A1.C20,A1.C21,A1.C22,A1.C23,A1.C24,A1.C25,A1.C26,A1.C27,A1.
C28,A1.C29,A1.C30,A1.C31,A1.C32,A1.C33,A1.C34,A1.C35,A1.C36,A1.C

        201,912            9       22,434.7    9.6     3.82    239.60 1246116920
Module: dmxt.exe
SELECT /*+ Q84417000 NO_EXPAND ROWID(A1) */ A1."ZXBMDM" C0,A1."F
MDM" C1,A1."FMGG" C2,A1."PFJE" C3,A1."JE" C4,A1."LSL" C5,A1."FMM
C" C6 FROM "YYGL"."MZ101_2" PX_GRANULE(0, BLOCK_RANGE, DYNAMIC)
 A1 WHERE (TO_CHAR(A1."SFRQ",'yyyy/mm/dd')=:B1 AND A1."JE">0 OR
TO_CHAR(A1."TFRQ",'yyyy/mm/dd')=:B2 AND A1."JE"<0) AND RTRIM(A1.
```

在很多情况下，并行也许并不是最好的选择，如果表并不大，并行反而会降低其执行速度。这个用户环境正是如此，询问用户，从未主动启用并行。

通过查询 DBA_TABLES 字典表可以获得 Degree 并行度的记录，并行度大于 1 的数据表在查询时会启用并行，但是注意事实还会有所不同，Degree 字段的类型及长度是 VARCHAR2(10)。所以注意，当使用类似如下查询时，可能无法获得返回值：

```
SQL> select table_name from dba_tables where degree='1' or degree='DEFAULT';
no rows selected
```

我们看一下 Degree 以及 Instances 的记录方式：

```
    SQL> select degree,length(degree) from dba_tables group by degree;
    DEGREE              LENGTH(DEGREE)
    -------------------- --------------
      DEFAULT                        10
            1                        10
    SQL>select instances,length(instances) from dba_tables group by instances;
    INSTANCES            LENGTH(INSTANCES)
    -------------------- -----------------
      DEFAULT                            10
            1                            10
            0                            10
```

Degree 和 Instances 实际上记录了 10 个字符，左端用空格补齐。在 dba_tables 的创建语句中，可以找到根本原因，以下是这两个字段的定义来源：

```
      lpad(decode(t.degree, 32767, 'DEFAULT', nvl(t.degree,1)),10),
      lpad(decode(t.instances, 32767, 'DEFAULT', nvl(t.instances,1)),10),
```

而需要注意的是，如果 Degree 设置为 DEFAULT，则默认数据库会对该表启用并行。最后找到相关的 SQL，从 AUTOTRACE 可以看到这些 SQL 的执行计划：

```
SQL> SELECT t1.fmdm, t1.fmmc, t1.sfdldm, t2.sfdlmc, t1.sfxldm, t3.sfxlmc, t1.fmdm,
  2         t1.fmmc, t1.dw, t1.dj, t1.fmshrm, NVL (t1.jeflag, '0'), t1.htbh,
  3         NVL (t1.zhflag, 0), t1.ybfl, t1.ybdm, '3'
  4    FROM sf007 t1, sf001 t2, sf006 t3
  5   WHERE t2.mzflag = '1'
  6     AND t1.sfdldm = t2.sfdldm
  7     AND t1.sfdldm = t3.sfdldm(+)
  8     AND t1.sfxldm = t3.sfxldm(+)
  9  /

1005 rows selected.

Execution Plan
----------------------------------------------------------
   0 SELECT STATEMENT Optimizer=CHOOSE (Cost=8 Card=16 Bytes=2192)
   1    0  HASH JOIN* (Cost=8 Card=16 Bytes=2192):Q91507003
   2    1    TABLE ACCESS* (FULL) OF 'SF001' (Cost=2 Card=4 Bytes=52):Q91507000
   3    1    HASH JOIN* (OUTER) (Cost=6 Card=1634 Bytes=202616):Q91507003
   4    3      TABLE ACCESS* (FULL) OF 'SF007' (Cost=5 Card=1634 Bytes=160132):Q91507001
   5    3      TABLE ACCESS* (FULL) OF 'SF006' (Cost=1 Card=409 Bytes=10634):Q91507002

   1 PARALLEL_TO_SERIAL            SELECT /*+ ORDERED NO_EXPAND USE_HASH(A2) SW
                                   AP_JOIN_INPUTS(A2) */ A1.C0,A2.C1,A1
   2 PARALLEL_FROM_SERIAL
   3 PARALLEL_COMBINED_WITH_PARENT
   4 PARALLEL_FROM_SERIAL
   5 PARALLEL_TO_PARALLEL          SELECT /*+ NO_EXPAND ROWID(A1) */ A1."SFDLDM
                                   " C0,A1."SFXLDM" C1,A1."SFXLMC" C2 F
```

查看涉及数据表的并行度，注意到其并行度被设置为 DEFAULT：

```
SQL> select table_name,degree from dba_tables where table_name='SF006';
TABLE_NAME                     DEGREE
------------------------------ ----------
SF006                           DEFAULT
```

将表的并行度修改为 1 后，问题得以解决：

```
SQL> alter table sf006 parallel 1;
Table altered.
```

这个问题给我们的启示是：并行并不总能够带来性能提升。

3．磁盘排序与临时文件

在 Oracle 10g 中，为了区分特定的对于临时文件的直接读写操作，Oracle 对 direct path read/write 进行了分离，将这类操作分列出来：

```
SQL> select event#,name,WAIT_CLASS
  2  from v$event_name where name like 'direct%';
    EVENT# NAME                             WAIT_CLASS
---------- -------------------------------- ------------------------------
       161 direct path read                 User I/O
       162 direct path read temp             User I/O
       163 direct path write                 User I/O
       164 direct path write temp            User I/O
```

可以看到，现在的 direct path read/write temp 就是单指对于临时文件的直接读写操作。结合 Oracle 10g 的一些特性，来进一步研究一下直接路径读/写与临时文件。

首先在一个 Session 中执行一个能够引发磁盘排序的查询：

```
SQL> select sid from v$mystat where rownum <2;
       SID
----------
       148
SQL> select a.table_name,b.object_name,b.object_type
  2  from t1 a ,t2 b where a.table_name = b.object_name
  3  order by b.object_name,b.object_type;
```

在另外 Session 查询相应等待事件：

```
SQL> select event,p1text,p1,p2text,p2,p3text,p3
  2  from v$session_wait_history where sid=148;
EVENT                        P1TEXT         P1 P2TEXT                P2 P3TEXT       P3
------------------------- ------------ ---------- --------- ---------- ------------ --
direct path read temp     file number          201 first dba     621872 block cnt    31
direct path read temp     file number        201 first dba     621872 block cnt    31
direct path read temp     file number        201 first dba      70232 block cnt    31
direct path read temp     file number        201 first dba      70232 block cnt    31
direct path read temp     file number        201 first dba     387706 block cnt    15
SQL*Net message to client driver id    1650815232 #bytes             1              0
direct path read temp     file number        201 first dba     409915 block cnt    31
direct path read temp     file number        201 first dba     409915 block cnt    31
direct path read temp     file number        201 first dba     198777 block cnt    16
direct path read temp     file number        201 first dba     198777 block cnt    16
10 rows selected
```

从以上输出可以看到最近的 10 次等待，direct path read temp 就是这个查询引起的磁盘排

序。注意这里的 file number 为 201。而实际上，通过 v$tempfile 来查询，临时文件的文件号仅为 1：

```
SQL> select file#,name from v$tempfile;
     FILE# NAME
---------- -----------------------------------------------------------------
         1 +ORADG/danaly/tempfile/temp.267.600173887
```

如果通过 10046 事件跟踪，也可以获得类似的结果：

```
WAIT #1: nam='direct path write' ela= 11 p1=201 p2=16584 p3=7
WAIT #1: nam='direct path write' ela= 2 p1=201 p2=16591 p3=7
WAIT #1: nam='direct path write' ela= 2 p1=201 p2=16598 p3=7
WAIT #1: nam='direct path write' ela= 8 p1=201 p2=16605 p3=1
WAIT #1: nam='direct path read' ela= 81 p1=201 p2=12937 p3=31
```

在 Oracle 文档中，File# 被定义为绝对文件号（The Absolute File Number）。这里的原因何在呢？研究这个问题的起因在于有朋友问起 V$TEMPSEG_USAGE 这个视图，可以从这个视图出发动手研究一下这个对象究竟来自何方。

查询 dba_objects 视图，发现 V$TEMPSEG_USAGE 原来是一个同义词。

```
SQL> select object_type from dba_objects where object_name='V$TEMPSEG_USAGE';
OBJECT_TYPE
------------------
SYNONYM
```

再追本溯源原来 V$TEMPSEG_USAGE 是 V_$SORT_USAGE 的同义词，也就是和 V$SORT_USAGE 同源。从 Oracle 9i 开始，Oracle 将 V$SORT_USAGE 视图从文档中移除了，因为这个名称有所歧义，容易使人误解仅记录排序内容，所以 V$TEMPSEG_USAGE 视图被引入，用于记录临时段的使用情况：

```
SQL> select * from dba_synonyms where synonym_name='V$TEMPSEG_USAGE';
OWNER      SYNONYM_NAME    TABLE_OWNE TABLE_NAME        DB_LINK
---------- --------------- ---------- ----------------- ----------
PUBLIC     V$TEMPSEG_USAGE SYS        V_$SORT_USAGE
```

如果再进一步，可以看到这个视图的构建语句:

```
SQL> SELECT view_definition FROM v$fixed_view_definition WHERE view_name='GV$SORT_USAGE';
VIEW_DEFINITION
--------------------------------------------------------------------------------
select x$ktsso.inst_id, username, username, ktssoses, ktssosno, prev_sql_addr, p
rev_hash_value, ktssotsn, decode(ktssocnt, 0, 'PERMANENT', 1, 'TEMPORARY'), deco
de(ktssosegt, 1, 'SORT', 2, 'HASH', 3, 'DATA', 4, 'INDEX', 5, 'LOB_DATA', 6, 'LO
B_INDEX' , 'UNDEFINED'), ktssofno, ktssobno, ktssoexts, ktssoblks, ktssorfno fro
m x$ktsso, v$session where ktssoses = v$session.saddr and ktssosno = v$session.s
erial#
```

格式化一下，v$sort_usage 的创建语句如下：

```
SELECT x$ktsso.inst_id, username, username, ktssoses, ktssosno, prev_sql_addr,
       prev_hash_value, ktssotsn,
       DECODE (ktssocnt, 0, 'PERMANENT', 1, 'TEMPORARY'),
       DECODE (ktssosegt,1, 'SORT',2, 'HASH',3, 'DATA',4, 'INDEX', 5, 'LOB_DATA',
               6, 'LOB_INDEX', 'UNDEFINED'),
       ktssofno, ktssobno, ktssoexts, ktssoblks, ktssorfno
  FROM x$ktsso, v$session
 WHERE ktssoses = v$session.saddr AND ktssosno = v$session.serial#;
```

注意到在 Oracle 文档中 SEGFILE#的定义为：

```
SEGFILE#   NUMBER  File number of initial extent
```

在视图中，这个字段来自 x$ktsso.ktssofno，也就是说这个字段实际上代表的是绝对文件号。那么这个绝对文件号如何与临时文件关联呢？能否与 V$TEMPFIE 中 file#字段关联呢？

再来看一下 V$TEMPFILE 的来源，V$TEMPFILE 由如下语句创建：

```
SELECT tf.inst_id, tf.tfnum, TO_NUMBER (tf.tfcrc_scn),
       TO_DATE (tf.tfcrc_tim, 'MM/DD/RR HH24:MI:SS', 'NLS_CALENDAR=Gregorian'),
       tf.tftsn, tf.tfrfn,
       DECODE (BITAND (tf.tfsta, 2), 0, 'OFFLINE', 2, 'ONLINE', 'UNKNOWN'),
       DECODE (BITAND (tf.tfsta, 12),0, 'DISABLED',4, 'READ ONLY',12, 'READ WRITE',
               'UNKNOWN'),
       fh.fhtmpfsz * tf.tfbsz, fh.fhtmpfsz, tf.tfcsz * tf.tfbsz, tf.tfbsz,fn.fnnam
  FROM x$kcctf tf, x$kccfn fn, x$kcvfhtmp fh
 WHERE fn.fnfno = tf.tfnum AND fn.fnfno = fh.htmpxfil AND tf.tffnh = fn.fnnum
   AND tf.tfdup != 0 AND fn.fntyp = 7 AND fn.fnnam IS NOT NULL
```

考察 x$kcctf 底层表，注意到 TFAFN（Temp File Absolute File Number）在这里存在：

```
SQL> desc x$kcctf
 Name                          Null?    Type
 ----------------------------- -------- --------------------
 ADDR                                   RAW(4)
 INDX                                   NUMBER
 INST_ID                                NUMBER
 TFNUM                                  NUMBER
 TFAFN                                  NUMBER
 TFCSZ                                  NUMBER
 TFBSZ                                  NUMBER
 TFSTA                                  NUMBER
 TFCRC_SCN                              VARCHAR2(16)
 TFCRC_TIM                              VARCHAR2(20)
 TFFNH                                  NUMBER
 TFFNT                                  NUMBER
```

```
TFDUP                                              NUMBER
TFTSN                                              NUMBER
TFTSI                                              NUMBER
TFRFN                                              NUMBER
TFPFT                                              NUMBER
```

而这个字段在构建 v$tempfile 时并未出现，所以不能通过 v$sort_usage 和 v$tempfile 直接关联绝对文件号。可以简单构建一个排序段使用，然后来继续研究一下：

```
SQL> select username,segtype,segfile#,segblk#,extents,segrfno# from v$sort_usage;
USERNAME SEGTYPE     SEGFILE#    SEGBLK#    EXTENTS   SEGRFNO#
-------- --------- ---------- ---------- ---------- ----------
SYS      LOB_DATA           9      18953          1          1
```

看到这里的 SEGFILE#=9，而在 v$tempfile 是找不到这个信息的：

```
SQL> select file#,rfile#,ts#,status,blocks from v$tempfile;
     FILE#     RFILE#        TS# STATUS      BLOCKS
---------- ---------- ---------- ------- ----------
         1          1          2 ONLINE       38400
```

但是可以从 x$kcctf 中获得这些信息，v$tempfile.file#实际上来自 x$kcctf.tfnum，是临时文件的文件号；而绝对文件号是 x$kcctf.tfafn，这个字段才可以与 v$sort_usage.segfile#关联：

```
SQL> select indx, tfnum, tfafn,tfcsz from x$kcctf;
      INDX      TFNUM      TFAFN      TFCSZ
---------- ---------- ---------- ----------
         0          1          9      38400
         1          2         10      12800
```

再进一步可以知道，实际上，为了分离临时文件号和数据文件号，Oracle 对临时文件的编号以 db_files 为起点，所以临时文件的绝对文件号应该等于 **db_files + file#**。

看前面引用到的 Oracle 10g 数据库的设置：

```
SQL> select indx,tfnum,tfafn,tfcsz from x$kcctf;
      INDX      TFNUM      TFAFN      TFCSZ
---------- ---------- ---------- ----------
         0          1        201       2560
```

db_files 参数的缺省值为 200：

```
SQL> show parameter db_files
NAME                                 TYPE        VALUE
------------------------------------ ----------- --------------
db_files                             integer     200
SQL> select file#,name from v$tempfile;
     FILE# NAME
---------- ----------------------------------------
         1 +ORADG/danaly/tempfile/temp.267.600173887
```

所以在 Oracle 文档中 v$tempfile.file#被定义为 The absolute file number 是不确切的。偶尔我们可能会在警报日志文件中看到类似如下的错误：

```
***
Corrupt block relative dba: 0x00c0008a (file 201, block 138)
Bad header found during buffer read
Data in bad block -
 type: 8 format: 2 rdba: 0x0140008a
 last change scn: 0x0000.431f8beb seq: 0x1 flg: 0x08
 consistency value in tail: 0x8beb0801
 check value in block header: 0x0, block checksum disabled
 spare1: 0x0, spare2: 0x0, spare3: 0x0
***
```

这里的 file 201 其实指的就是临时文件。以上的整个过程更主要的是说明一个思路，供大家在解决或研究问题时参考。

4．串行全表扫描 – Serial Table Scan

在 Oracle 11g 之前，全表扫描使用 db file scattered read 的方式，将表中的数据块离散的读到 Buffer Cache 之后，供用户访问和使用，但是如果全表访问的表非常大，则有可能占用大量的 Buffer Cache 内存，这会导致 Buffer Cache 中其他数据被老化和挤出内存，而且在这一系列的读取操作中，Oracle 引擎需要去判断每一个数据块是否已经存在于内存中，然后还要去请求内存空间，不断使用 Cache Buffer Chain 和 Cache Buffer Lru Chain 两个 Latch 进行判断,在某种程度上会加剧 Latch 竞争,如果全表访问的数据只是偶尔个别的访问,则占据大量 Buffer Cache 就显得过于昂贵,在 Oracle Database 11g 中,一种被称为**串行全表扫描(Serial Table Scan)**的技术被引入,该特性根据数据块的设置和统计信息等,自动决定是采用 Direct Path Read 绕过 SGA,还是采用常规方式读取,因为这种自动选择,这一特性又被称为**自适应直接读(Adaptive Direct Read)** .这种方式的好处是可以降低 Buffer Cache 的竞争,但是每次都要发生物理读,而且在读取之前可能需要触发检查点,避免读到旧的映像.

以下通过一个测试来描述一下以上的特性,测试版本为 Oracle 11.2.0.2 ,首先创建一个测试表,插入足够的记录:

```
SQL> create table eygle as select * from dba_objects;


表已创建。


SQL> insert into eygle select * from eygle;
已创建 13334 行。


SQL> /
已创建 26668 行。

```

```
..............
SQL> commit;
提交完成。
SQL> exec dbms_stats.gather_table_stats(user,'EYGLE');

PL/SQL 过程已成功完成。
SQL> select blocks,num_rows from dba_tables where table_name='EYGLE';
    BLOCKS   NUM_ROWS
---------- ----------
     12149     880044
```

执行 Flush Buffer_Cache,清理缓存,再次插入部分数据,创建内存脏数据:

```
SQL> alter system flush buffer_cache;

系统已更改。

SQL> insert into eygle select * from dba_objects;

已创建 13334 行。

SQL> commit;

提交完成。
```

使用 10046 事件跟踪一次全表扫描:

```
SQL> alter session set events '10046 trace name context forever,level 12';

会话已更改。

SQL> select count(*) from eygle;
  COUNT(*)
----------
    893378

SQL> alter session set events '10046 trace name context off';

会话已更改。
```

获得跟踪文件:

```
SQL> select value from v$diag_info where name='Default Trace File';
VALUE
--------------------------------------------------------------------------
D:\ORACLE\diag\rdbms\ora11g\ora11g\trace\ora11g_ora_3068.trc
```

以下是跟踪文件的部分摘录,我们可以看到Oracle引擎执行了检查点,然后通过直接路径读取访问数据:

```
=====================
PARSING IN CURSOR #824687444 len=26 dep=0 uid=34 oct=3 lid=34 tim=982908005 hv=2664040539 ad='2a59f65c' sqlid='8ufz53kgcn22v'
select count(*) from eygle
END OF STMT
PARSE #824687444:c=0,e=1009,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=1,plh=3602634261,tim=982907999
EXEC #824687444:c=0,e=63,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,plh=3602634261,tim=982908159
WAIT #824687444: nam='SQL*Net message to client' ela= 10 driver id=1111838976 #bytes=1 p3=0 obj#=13753 tim=982908255
WAIT #824687444: nam='reliable message' ela= 453 channel context=787207128 channel handle=787161240 broadcast message=788173128 obj#=13753 tim=982908906
WAIT #824687444: nam='enq: KO - fast object checkpoint' ela= 19841 name|mode=1263468550 2=65566 0=1 obj#=13753 tim=982928842
WAIT #824687444: nam='Disk file operations I/O' ela= 1504 FileOperation=2 fileno=4 filetype=2 obj#=13753 tim=982930548
WAIT #824687444: nam='direct path read' ela= 197 file number=4 first dba=177 block cnt=15 obj#=13789 tim=982986667
WAIT #824687444: nam='direct path read' ela= 10410 file number=4 first dba=193 block cnt=15 obj#=13789 tim=982997376
WAIT #824687444: nam='direct path read' ela= 1487 file number=4 first dba=209 block cnt=15 obj#=13789 tim=982999158
WAIT #824687444: nam='direct path read' ela= 1431 file number=4 first dba=225 block cnt=15 obj#=13789 tim=983000833
WAIT #824687444: nam='direct path read' ela= 1372 file number=4 first dba=241 block cnt=15 obj#=13789 tim=983002616
WAIT #824687444: nam='direct path read' ela= 15190 file number=4 first dba=258 block cnt=126 obj#=13789 tim=983019131
WAIT #824687444: nam='direct path read' ela= 32791 file number=4 first dba=386 block cnt=126 obj#=13789 tim=983053651
WAIT #824687444: nam='direct path read' ela= 28783 file number=4 first dba=642 block cnt=126 obj#=13789 tim=983087063
WAIT #824687444: nam='direct path read' ela= 22889 file number=4 first dba=770 block cnt=126 obj#=13789 tim=983111700
WAIT #824687444: nam='direct path read' ela= 10091 file number=4 first dba=12288 block cnt=128 obj#=13789 tim=984893761
FETCH #824687444:c=170244,e=1987190,p=11641,cr=11650,cu=0,mis=0,r=1,dep=0,og=1,plh=3602634261,tim=984895492
STAT #824687444 id=1 cnt=1 pid=0 pos=1 obj=0 op='SORT AGGREGATE (cr=11650 pr=11641 pw=0 time=1987186 us)'
STAT #824687444 id=2 cnt=893378 pid=1 pos=1 obj=13789 op='TABLE ACCESS FULL EYGLE (cr=11650 pr=11641 pw=0 time=4763412 us cost=3301 size=0 card=880044)'
WAIT #824687444: nam='SQL*Net message from client' ela= 218 driver id=1111838976 #bytes=1 p3=0 obj#=13789 tim=984896026
FETCH #824687444:c=0,e=6,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=0,plh=3602634261,tim=984896108
WAIT #824687444: nam='SQL*Net message to client' ela= 7 driver id=1111838976 #bytes=1 p3=0 obj#=13789 tim=984896158
=====================
```

使用tkprof格式化之后会获得如下主要信息:

```
********************************************************************************
SQL ID: 8ufz53kgcn22v Plan Hash: 3602634261

select count(*) from eygle
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        2      0.17       1.98      11641      11650          0           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.17       1.98      11641      11650          0           1

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 34
Number of plan statistics captured: 1

Rows (1st) Rows (avg) Rows (max)  Row Source Operation
---------- ---------- ----------  ---------------------------------------------------
         1          1          1  SORT AGGREGATE (cr=11650 pr=11641 pw=0 time=1987186 us)
    893378     893378     893378   TABLE ACCESS FULL EYGLE (cr=11650 pr=11641 pw=0
time=4763412 us cost=3301 size=0 card=880044)



Elapsed times include waiting on following events:
  Event waited on                             Times   Max. Wait  Total Waited
  ----------------------------------------   Waited  ----------  ------------
  SQL*Net message to client                       2        0.00          0.00
  reliable message                                1        0.00          0.00
  enq: KO - fast object checkpoint                1        0.01          0.01
  Disk file operations I/O                        1        0.00          0.00
  direct path read                               81        0.09          1.67
  SQL*Net message from client                     2        1.77          1.77
********************************************************************************
```

以下输出显示,串行表扫描以物理读方式执行,每次执行该查询,产生同样的物理读:

```
SQL> connect eygle/eygle
已连接。
SQL> select a.name,b.value from v$statname a,v$mystat b where a.statistic#=b.statistic#
and a.name='physical reads';
```

```
NAME                                                              VALUE
---------------------------------------------------------------- ----------
physical reads                                                         0
SQL> select count(*) from eygle;
  COUNT(*)
----------
    893378
SQL> select a.name,b.value from v$statname a,v$mystat b where a.statistic#=b.statistic#
and a.name='physical reads';
NAME                                                              VALUE
---------------------------------------------------------------- ----------
physical reads                                                     11641
SQL> connect eygle/eygle
已连接。
SQL> select a.name,b.value from v$statname a,v$mystat b where a.statistic#=b.statistic#
and a.name='physical reads';
NAME                                                              VALUE
---------------------------------------------------------------- ----------
physical reads                                                         0
SQL> select count(*) from eygle;
  COUNT(*)
----------
    893378
SQL> select a.name,b.value from v$statname a,v$mystat b where a.statistic#=b.statistic#
and a.name='physical reads';
NAME                                                              VALUE
---------------------------------------------------------------- ----------
physical reads                                                     11641
```

使用串行全表扫描和多个因素有关,首先全表访问的数据表需要至少超过 5 倍的 **_small_table_threshold 设置,因为通常小表的全表访问并不会对 Buffer Cache 产生过大的冲击.这个隐含参数的缺省值如下:**

```
SQL> SELECT x.ksppinm NAME, y.ksppstvl VALUE, x.ksppdesc describ
  2    FROM SYS.x$ksppi x, SYS.x$ksppcv y
  3   WHERE x.indx = y.indx AND x.ksppinm LIKE '%&par%';
Enter value for par: small_table_
old   3:  WHERE x.indx = y.indx AND x.ksppinm LIKE '%&par%'
new   3:  WHERE x.indx = y.indx AND x.ksppinm LIKE '%small_table_%'
```

```
NAME                          VALUE              DESCRIB
-----------------------------                                                 --------------------
----------------------------------------------------
_small_table_threshold        803                lower threshold level of table size for direct reads
```

Oracle支持通过 10949 事件禁用串行全表扫描(但是当表大于 5 倍的 Buffer Cache 时,则不允许禁用,只能采用直接路径读):

```
[oracle@wybm ~]$ oerr ORA 10949
10949, 00000, "Disable autotune direct path read for full table scan"
// *Cause:
// *Action:  Disable autotune direct path read for serial full table scan.
```

再看如下测试:

```
SQL> alter system flush buffer_cache;

系统已更改。

SQL> alter session set events '10046 trace name context forever,level 12';

会话已更改。

SQL> alter session set events '10949 trace name context forever, level 1';

会话已更改。

SQL> select count(*) from eygle;

  COUNT(*)
----------
    893378

SQL> alter session set events '10046 trace name context off';

会话已更改。
```

通过跟踪文件可以获得详细的,熟悉的输出:

```
*** 2011-07-01 11:41:50.785
WAIT #247955564: nam='db file scattered read' ela= 1703 file#=4 block#=6530 blocks=126 obj#=13789
tim=7122074156
WAIT #247955564: nam='db file scattered read' ela= 1606 file#=4 block#=6658 blocks=126 obj#=13789
tim=7122077945
```

```
WAIT #247955564: nam='db file scattered read' ela= 1675 file#=4 block#=6786 blocks=126 obj#=13789
tim=7122081812
WAIT #247955564: nam='db file scattered read' ela= 1650 file#=4 block#=6914 blocks=126 obj#=13789
tim=7122086829
WAIT #247955564: nam='db file scattered read' ela= 1950 file#=4 block#=7042 blocks=126 obj#=13789
tim=7122091857
WAIT #247955564: nam='db file scattered read' ela= 1885 file#=4 block#=7170 blocks=126 obj#=13789
tim=7122096332
WAIT #247955564: nam='db file scattered read' ela= 1720 file#=4 block#=8836 blocks=128 obj#=13789
tim=7122152331
WAIT #247955564: nam='db file scattered read' ela= 1713 file#=4 block#=8964 blocks=128 obj#=13789
tim=7122157236
WAIT #247955564: nam='db file scattered read' ela= 1688 file#=4 block#=9092 blocks=128 obj#=13789
tim=7122161418
WAIT #247955564: nam='db file scattered read' ela= 1845 file#=4 block#=12160 blocks=128 obj#=13789
tim=7122244250
WAIT #247955564: nam='db file scattered read' ela= 1635 file#=4 block#=12288 blocks=128 obj#=13789
tim=7122247817
FETCH
#247955564:c=290417,e=389668,p=11648,cr=11659,cu=1,mis=0,r=1,dep=0,og=1,plh=3602634261,tim=7122249720
STAT #247955564 id=1 cnt=1 pid=0 pos=1 obj=0 op='SORT AGGREGATE (cr=11659 pr=11648 pw=0 time=389667 us)'
STAT #247955564 id=2 cnt=893378 pid=1 pos=1 obj=13789 op='TABLE ACCESS FULL EYGLE (cr=11659 pr=11648 pw=0
time=4807540 us cost=3024 size=0 card=853376)'
WAIT #247955564: nam='SQL*Net message from client' ela= 153 driver id=1111838976 #bytes=1 p3=0 obj#=13789
tim=7122250022
FETCH #247955564:c=0,e=5,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=0,plh=3602634261,tim=7122250077
WAIT #247955564: nam='SQL*Net message to client' ela= 5 driver id=1111838976 #bytes=1 p3=0 obj#=13789
tim=7122250117
```

格式化输出的跟踪文件,获得如下信息:

```
********************************************************************************
SQL ID: 8ufz53kgcn22v Plan Hash: 3602634261

select count(*) from eygle

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
```

```
Fetch        2      0.29       0.38      11648      11659          1           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.29       0.38      11648      11659          1           1

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 34
Number of plan statistics captured: 1

Rows (1st) Rows (avg) Rows (max)  Row Source Operation
---------- ---------- ----------  ---------------------------------------------------
         1          1          1  SORT AGGREGATE (cr=11659 pr=11648 pw=0 time=389667 us)
    893378       893378       893378    TABLE ACCESS FULL EYGLE (cr=11659 pr=11648 pw=0
time=4807540 us cost=3024 size=0 card=853376)



Elapsed times include waiting on following events:
  Event waited on                             Times   Max. Wait  Total Waited
  ----------------------------------------   Waited  ----------  ------------
  SQL*Net message to client                       2        0.00          0.00
  db file sequential read                         4        0.00          0.00
  db file scattered read                        109        0.00          0.16
  SQL*Net message from client                     2        2.94          2.94
********************************************************************************
```

10949 事件可以禁用串行表扫描,而另外一个隐含参数_serial_direct_read 则用于启用串行直接路径读:

```
NAME                          VALUE                DESCRIB
----------------------------- -------------------- --------------------------------
_serial_direct_read           auto                 enable direct read in serial
```

通过前面的测试可以看到串行表扫描可以提升全表扫描的性能,但是并非总是如此,尤其是当数据可以 Cache 在内存中被反复访问时,无疑 Scattered Read 更有优势,频繁的物理访问对于 IO 密集系统无疑将是灾难,所以这一技术需要在具体环境中根据判断去应用.

以下是一个来自于实践的相关案例,通过 AWR 报告可以看到,这个数据库服务器是 64 位 Windows 平台,仅有 2G 内存,数据库版本为 11.1.0.7,采样时间一小时:

DB Name	DB Id	Instance	Inst num	Startup Time	Release	RAC
GVDB	2466339014	gvdb	1	25-3月 -11 03:03	11.1.0.7.0	NO

Host Name	Platform	CPUs	Cores	Sockets	Memory (GB)
BJCBDSERVER	Microsoft Windows x86 64-bit	4	1		2.00

	Snap Id	Snap Time	Sessions	Cursors/Session
Begin Snap:	3087	29-6月 -11 09:00:45	79	1.8
End Snap:	3088	29-6月 -11 10:00:55	89	1.9
Elapsed:		60.17 (mins)		
DB Time:		73.06 (mins)		

下面我们来看一下这个数据库的内存配置及负载概要信息.以下数据显示,数据库的 Buffer Cache 仅有 192M,Shared Pool 为 312M,而物理度每秒有 8169 次,这说明数据库的内存配置是较低的:

Cache Sizes

	Begin	End		
Buffer Cache:	192M	192M	Std Block Size:	8K
Shared Pool Size:	312M	312M	Log Buffer:	6,440K

Load Profile

	Per Second	Per Transaction	Per Exec	Per Call
DB Time(s):	1.2	1.0	0.01	0.03
DB CPU(s):	0.3	0.3	0.00	0.01
Redo size:	3,139.5	2,471.9		
Logical reads:	10,430.5	8,212.6		
Block changes:	18.9	14.9		
Physical reads:	8,169.9	6,432.7		
Physical writes:	0.9	0.7		

进一步的,等待事件会帮助我们提供更多的信息,在以下 Top 5 事件中,Direct Path Read 占据了第一位,占 DB Time 的 58.13%,非常显著,那么这一事件是如何出现的呢?

这就是我们前面提到的串行表扫描:

Top 5 Timed Foreground Events

Event	Waits	Time(s)	Avg wait (ms)	% DB time	Wait Class
direct path read	789,460	2,548	3	58.13	User I/O
DB CPU		1,170		26.69	
log file sync	4,297	172	40	3.92	Commit
SQL*Net more data from dblink	17,712	168	10	3.84	Network
db file sequential read	8,719	78	9	1.79	User I/O

通过 ASH 报告可以抓取一些 SQL 的执行信息,以下报告显示,全表访问正是通过 Direct Path Read 方式实现的.

Top SQL with Top Events

SQL ID	Planhash	Sampled # of Executions	% Activity	Event	% Event	Top Row Source	% RwSrc	SQL Text
0amh8a9gapck3	3799811152	138	17.28	direct path read	12.46	TABLE ACCESS - FULL	12.46	select TKT_TICKET_NO, TKT_SEQ_...
				CPU + Wait for CPU	4.82	TABLE ACCESS - FULL	4.82	
92j2h92crxz9n	2419806026	142	14.58	direct path read	9.73	TABLE ACCESS - FULL	9.73	select BPM_BOOKING_ID, BPM_TIC...
				CPU + Wait for CPU	4.82	TABLE ACCESS - FULL	4.82	
dx5tvp56ug13b	3749003550	119	8.16	direct path read	6.01	TABLE ACCESS - FULL	6.01	select BKG_BOOKING_ID, to_char...
				CPU + Wait for CPU	2.15	TABLE ACCESS - FULL	2.15	
32z4qspczrf5n	3799811152	43	5.81	direct path read	4.27	TABLE ACCESS - FULL	4.27	select TKT_TICKET_NO, TKT_SEQ_...
				CPU + Wait for CPU	1.54	TABLE ACCESS - FULL	1.54	
ccc6ma0dfx6mf	2419806026	42	4.75	direct path read	3.24	TABLE ACCESS - FULL	3.24	select BPM_BOOKING_ID, BPM_TIC...
				CPU + Wait for CPU	1.51	TABLE ACCESS - FULL	1.51	

Back to Top SQL
Back to Top

Top SQL with Top Row Sources

SQL ID	PlanHash	Sampled # of Executions	% Activity	Row Source	% RwSrc	Top Event	% Event	SQL Text
0amh8a9gapck3	3799811152	138	17.28	TABLE ACCESS - FULL	17.28	direct path read	12.46	select TKT_TICKET_NO, TKT_SEQ_...
92j2h92crxz9n	2419806026	142	14.58	TABLE ACCESS - FULL	14.58	direct path read	9.73	select BPM_BOOKING_ID, BPM_TIC...
dx5tvp56ug13b	3749003550	119	8.16	TABLE ACCESS - FULL	8.16	direct path read	6.01	select BKG_BOOKING_ID, to_char...
32z4qspczrf5n	3799811152	43	5.81	TABLE ACCESS - FULL	5.81	direct path read	4.27	select TKT_TICKET_NO, TKT_SEQ_...
ccc6ma0dfx6mf	2419806026	42	4.75	TABLE ACCESS - FULL	4.75	direct path read	3.24	select BPM_BOOKING_ID, BPM_TIC...

在客户的系统中,通过添加适当的索引,可以解决这些全表扫描的问题，进一步缩减了 Direct Path Read 的等待。如果不能创建合适的索引，则对于确定表的反复扫描，也应当通过 Cache 来降低物理读。

9.5.4　日志文件相关等待

第 6 章已经详细介绍了重做的相关知识，Redo 对于数据库来说非常重要，有一系列等待事件和日志相关，通过 v$event_name 视图可以找到这些等待事件：

```
SQL> select name from v$event_name where name like '%log%';
NAME
----------------------------------------------------------------
log switch/archive
```

```
log file sequential read
log file single write
log file parallel write
log buffer space
log file switch (checkpoint incomplete)
log file switch (archiving needed)
log file switch (clearing log file)
switch logfile command
log file switch completion
log file sync
STREAMS capture process waiting for archive log
rfrxptarcurlog

13 rows selected.
```

下面摘录几个重要事件进行详细介绍。

1. log file switch（日志文件切换）

log file switch 当日志文件发生切换时出现，在数据库进行日志切换时，后台进程 LGWR 需要关闭当前日志组，切换并打开下一个日志组，在这个切换过程中，数据库的所有 DML 操作都处于停顿状态，直至这个切换完成。

log file switch 主要包含两个子事件 log file switch（archiving needed）和 log file switch（checkpoint incomplete）。

（1）log file switch（archiving needed），即日志切换（需要归档），这个等待事件出现时通常是因为日志组循环写满以后，在需要覆盖先前日志时，发现日志归档尚未完成，出现该等待。由于 Redo 不能写出，该等待出现时，数据库将陷于停顿状态。

出现该等待，可能表示 I/O 存在问题、归档进程写出缓慢，也有可能是日志组设置不合理等原因导致。针对不同原因，可以考虑采用的解决办法有：

- 可以考虑增大日志文件和增加日志组；
- 移动归档文件到快速磁盘；
- 调整 log_archive_max_processes 参数等。

（2）log file switch（checkpoint incomplete），即日志切换（检查点未完成）。当所有的日志组都写满之后，LGWR 试图覆盖某个日志文件，如果这时数据库没有完成写出由这个日志文件所保护的脏数据时（检查点未完成），该等待事件出现。该等待出现时，数据库同样将陷于停顿状态。

同时警告日志文件中会记录如下信息：

```
Fri Nov 18 14:26:57 2005
Thread 1 cannot allocate new log, sequence 7239
Checkpoint not complete
  Current log# 5 seq# 7238 mem# 0: /opt/oracle/oradata/hsmkt/redo05.log
```

该等待事件通常表示 DBWR 写出速度太慢或者 I/O 存在问题。为解决该问题，用户可能

需要考虑增加额外的 DBWR 或者增加日志组或日志文件大小。log file switch 引起的等待都是非常重要的，如果出现就应该引起重视，并由 DBA 介入进行及时处理。

2. log file sync（日志文件同步）

当一个用户提交或回滚数据时，LGWR 将会话期的重做由 Log Buffer 写入到重做日志中，LGWR 完成任务以后会通知用户进程。日志文件同步等待（Log File Sync）就是指进程等待 LGWR 写完成这个过程；对于回滚操作，该事件记录从用户发出 rollback 命令到回滚完成的时间。

如果该等待过多，可能说明 LGWR 的写出效率低下，或者系统提交过于频繁。针对该问题，可以关注 log file parallel write 等待事件，或者通过 user commits,user rollback 等统计信息观察提交或回滚次数。

可能的解决方案主要有：

- 提高 LGWR 性能，尽量使用快速磁盘，不要把 redo log file 存放在 RAID5 的磁盘上；
- 使用批量提交；
- 适当使用 NOLOGGING/UNRECOVERABLE 等选项。

可以通过如下公式计算平均 Redo 写大小：

avg.redo write size = (Redo block written/redo writes)*512 bytes

如果系统产生 Redo 很多，而每次写的较少，一般说明 LGWR 被过于频繁地激活了。可能导致过多的 Redo 相关 Latch 的竞争，而且 Oracle 可能无法有效地使用 piggyback 的功能。从一个 Statspack 报告中提取一些数据来研究一下这个问题。

Report 概要信息如下：

```
DB Name         DB Id    Instance     Inst Num Release     OPS Host
------------ ----------- ------------ -------- ----------- --- ------------
DB          1222010599  oracle              1 9.1.7.4.5   NO  sun
                Snap Id     Snap Time      Sessions
                ------- ------------------ --------
 Begin Snap:       3473 13-Oct-04 13:43:00      540
   End Snap:       3475 13-Oct-04 14:07:28      540
    Elapsed:                 24.47 (mins)
Cache Sizes
~~~~~~~~~~~
           db_block_buffers:     102400          log_buffer:   20971520
              db_block_size:       8192    shared_pool_size:       600M
Load Profile
~~~~~~~~~~~~                            Per Second       Per Transaction
                                   ---------------      ---------------
                  Redo size:             28,459.11             2,852.03
```

等待事件如下：

```
 Event                                Waits    Timeouts  Time (cs)    (ms)    /txn
```

```
--------------------------- ------------ ---------- ---------- ------ ------
log file sync                      14,466           2        4,150       3    1.0
db file sequential read             17,202           0        2,869       2    1.2
latch free                          24,841      13,489        2,072       1    1.7
direct path write                      121           0        1,455     120    0.0
db file parallel write               1,314           0        1,383      11    0.1
log file sequential read             1,540           0           63       0    0.1
log file switch completion               1           0            3      30    0.0
refresh controlfile command             23           0            1       0    0.0
LGWR wait for redo copy                 46           0            0       0    0.0
log file single write                    4           0            0       0    0.0
```

注意以上输出信息，这里 log file sync 和 db file parallel write 等等待事件同时出现，那么可能的一个原因是 I/O 竞争导致了性能问题，实际用户环境正是日志文件和数据文件同时存放在 RAID5 的磁盘上，存在性能问题需要调整。

统计信息如下：

```
 Statistic                                     Total     per Second     per Trans
--------------------------------- ---------------- ------------ ------------
……
redo blocks written                           93,853           63.9           6.4
redo buffer allocation retries                     1            0.0           0.0
redo entries                                 135,837           92.5           9.3
redo log space requests                            1            0.0           0.0
redo log space wait time                           3            0.0           0.0
redo ordering marks                                0            0.0           0.0
redo size                                 41,776,508       28,459.1       2,852.0
redo synch time                                4,174            2.8           0.3
redo synch writes                             14,198            9.7           1.0
redo wastage                               4,769,200        3,249.8         325.6
redo write time                                3,698            2.5           0.3
redo writer latching time                          0            0.0           0.0
redo writes                                   14,572            9.9           1.0
……
sorts (disk)                                       4            0.0           0.0
sorts (memory)                               179,856          122.5          12.3
sorts (rows)                               2,750,980        1,874.0         187.8
……
transaction rollbacks                             36            0.0           0.0
transaction tables consistent rea                  0            0.0           0.0
transaction tables consistent rea                  0            0.0           0.0
```

```
user calls                                 1,390,718       947.4          94.9
user commits                                  14,136          9.6            1.0
user rollbacks                                   512         0.4            0.0
write clones created in backgroun                  0         0.0            0.0
write clones created in foregroun                 11         0.0            0.0
          -------------------------------------------------------------
```

根据统计信息可以计算平均日志写大小：

avg.redo write size = (Redo block written/redo writes)*512 bytes

= (93,853 / 14,572)*512

= 3KB

这个平均值过小了，说明系统的提交过于频繁。从以上的统计信息中，可以看到平均每秒数据库的提交数量是 9.6 次。如果可能，在设计应用时应该选择合适的提交批量，从而提高数据库的效率。

```
 Latch Sleep breakdown for DB: DPSHDB  Instance: dpshdb  Snaps: 3473 -3475
-> ordered by misses desc
                                Get                                   Spin &
Latch Name                     Requests        Misses      Sleeps Sleeps 1->4
-------------------------- -------------- ----------- ----------- ------------
row cache objects              12,257,850     113,299          64 113235/64/0/0/0
shared pool                     3,690,715      60,279      15,857 52484/588/6546/661/0
library cache                   4,912,465      29,454       8,876 23823/2682/2733/216/0
cache buffers chains           10,314,526       2,856          33 2823/33/0/0/0
redo writing                       76,550         937              1 936/1/0/0/0
session idle bit                2,871,949         225           1 224/1/0/0/0
messages                          107,950         159           2 157/2/0/0/0
session allocation                184,386          44           6 38/6/0/0/0
checkpoint queue latch             96,583           1           1 0/1/0/0/0
          -------------------------------------------------------------
```

由于过度频繁的提交，LGWR 过度频繁的激活，看到这里出现了 redo writing 的 latch 竞争。

以下是一则 ASH 报告中显示的 Log File Sync 等待信息，注意到其 Parameter 1 是 Buffer#，Parameter 2 代表 Sync SCN，也就是同步的 SCN。Log File Sync 以 SCN 为节点，以 Buffer 号为起始，不断将 Log Buffer 的内容写出到日志文件上来：

Top Event P1/P2/P3 Values

Event	% Event	P1 Value, P2 Value, P3 Value	% Activity	Parameter 1	Parameter 2	Parameter 3
db file sequential read	10.81	"7","809389","1"	1.35	file#	block#	blocks
		"8","1749501","1"	1.35			
		"9","350240","1"	1.35			
log file sync	10.81	"2268","1849513965","0"	1.35	buffer#	sync scn	NOT DEFINED
		"5060","1849596977","0"	1.35			
		"5982","1849465171","0"	1.35			
log file parallel write	8.11	"1","2","1"	5.41	files	blocks	requests
		"1","3","1"	1.35			
		"1","49","1"	1.35			

3. 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（log switch）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。在 log switch 的过程中，LGWR 需要改写日志文件头，有时可以观察到该等待事件的增加：

```
SQL> select event,time_waited from v$system_event where event='log file single write';
EVENT                                                            TIME_WAITED
---------------------------------------------------------------- -----------
log file single write                                                   2848
SQL> alter system switch logfile;
System altered.
SQL> alter system switch logfile;
System altered.
SQL> select event,time_waited from v$system_event where event='log file single write';
EVENT                                                            TIME_WAITED
---------------------------------------------------------------- -----------
log file single write                                                   2853
SQL> alter system switch logfile;
System altered.
SQL>  select event,time_waited from v$system_event where event='log file single write';
EVENT                                                            TIME_WAITED
---------------------------------------------------------------- -----------
log file single write                                                   2855
```

4. log file parallel write

从 log buffer 写 Redo 记录到日志文件，主要指常规写操作（相对于 log file sync）。如果每个日志组存在多个组成员，当 flush log buffer 时，写操作是并行的，这时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。

5. log Buffer Space-日志缓冲空间

当数据库产生日志的速度比 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，或者增加日志缓冲区的大小。

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，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，尽量使用 RAID10 而不是 RAID5 磁盘来存储日志文件。以下是一个 log buffer 存在问题的 Statspack Top5 等待事件的系统：

```
Top 5 Wait Events
~~~~~~~~~~~~~~~                  Wait     % Total
Event                     Waits  Time (cs)   Wt Time
-------------------------------------------- ------------ ------------ -------
log file parallel write      1,436,993    1,102,188   10.80
log buffer space               16,698      873,203    9.56
log file sync              1,413,374      654,587    6.42
control file parallel write    329,777      510,078    5.00
db file scattered read       425,578      132,537    1.30
-------------------------------------------------------------
```

Log Buffer Space 等待事件出现时，数据库将陷于停顿状态，所有和日志生成相关的操作全部不能进行，所以这个等待事件应该引起充分的重视。

8.5.5 Enqueue（队列等待）

Enqueue 是一种保护共享资源的锁定机制。该锁定机制保护共享资源，以避免因并发操作而损坏数据，比如通过锁定保护一行记录，避免多个用户同时更新。Enqueue 采用排队机制，即 FIFO（先进先出）来控制资源的使用。

在 Oracle 10g 之前，Enqueue 事件是一组锁定事件的集合，如果数据库中这个等待事件比较显著，我们还需要进一步来追踪是哪一个类别的锁定引发了数据库等待。

从 Oracle 10g 开始，Oracle 对于队列等待进行了细分，V$EVENT_NAME 视图中可以查询这些细分后的等待事件，简要摘录几个示例如下：

```
SQL> select name,wait_class
  2  from v$event_name where name like '%enq%';
NAME                                                WAIT_CLASS
--------------------------------------------------- -------------------------
```

```
enq: PW - flush prewarm buffers                    Application
enq: RO - contention                               Application
enq: RO - fast object reuse                        Application
enq: KO - fast object checkpoint                   Application
enq: TM - contention                               Application
enq: ST - contention                               Configuration
enq: HW - contention                               Configuration
enq: SS - contention                               Configuration
enq: TX - row lock contention                      Application
enq: TX - allocate ITL entry                       Configuration
enq: TX - index contention                         Concurrency
……
```

Oracle 的锁按照类型可以分为排他锁（Exclusive，缩写为 X）与共享锁（Share，缩写为 S），或者是两者的组合锁。排他锁（X）也被称为独占锁，在排他锁释放之前，一个对象上不能施加任何其他类型的锁定；而共享锁（S）在释放之前，对象上还可以继续加其他类型的共享锁，但是不能加排他锁。

如果按照事务的类型划分，又可以将锁定划分为 DML 锁、DDL 锁已经内存锁（也即通常所说的 Latch）。Oracle 在数据库内部用 Enqueue 等待来记录锁定，通过 Latch Free 等待事件来记录闩。Enqueue 等待常见的有 ST、HW、TX、TM 等，下面进行择要介绍。

1．最重要的锁定：TM 与 TX 锁

对于数据库来说，最常见的锁定类型是 TM 以及 TX 锁定。

TX 锁通常被称为事务锁，当一个事务开始时，如执行 INSERT/DELETE/UPDATE/ MERGE 等操作或者使用 SELECT … FOR UPDATE 语句进行查询时，会首先获取事务锁，直到该事务结束。Oracle 的 TX 锁定是在行级获得的，每个数据行上都存在一个锁定位（lb-Lock Byte），用于判断该记录是否被锁定，同时在每个数据块的头部（Header）存在一个被称为 ITL 的数据结构，用于记录事务信息等，当需要修改数据时，首先需要获得回滚段空间用于存储前镜像信息，然后这个事务信息同样被记录在 ITL 上，通过 ITL 可以将回滚信息和数据块关联起来，所以说 Oracle 的行级锁定是在数据块上获得的，行级锁只有排他锁没有共享模式。

TM 锁通常称为表级锁，可以通过手工发出 lock 命令获得，或者通过 DML 操作以及 SELECT FOR UPDATE 获得，表级锁可以防止其他进程对表加 X 排他锁，防止在对数据修改时，其他任务通过 DDL 来修改表结构或者执行 Truncate、Drop 表等操作。可以通过 V$LOCK 视图来观察锁定信息，其中 TYPE 字段表示锁定类型。对于 TM 所 LMODE 字段又代表了不同级别的 TM 锁，这些级别包括 2 - row-S (SS)、3 - row-X (SX)、4 - share (S)、5 - S/Row-X (SSX) 和 6 - exclusive (X)。

当执行 DML 操作时，首先加 TM 锁，如果能获得锁定，则继续加 TX 事务锁。在一个会话中，一般只存在一个 TX 事务锁，在提交或回滚之前，该会话的所有 DML 操作都属于一个事务，使用一个回滚段，占用一个回滚段事务槽（Slot）。

以下通过 SCOTT 用户锁定一行记录，暂时不要提交：

```
SQL> update emp set sal=4000 where empno=7788;

1 row updated.
```

在另外 Session 通过 V$LOCK 视图可以看到相关的锁定信息：

```
SQL> select sid,username from v$session where username='SCOTT';
       SID USERNAME
---------- ------------------------------
       159 SCOTT
SQL> select * from v$lock where sid=159;
ADDR     KADDR     SID TY        ID1        ID2  LMODE    REQUEST      CTIME      BLOCK
-------- -------- ---- -- ---------- ---------- ------ ---------- ---------- ----------
5EE798EC 5EE79904  159 TM      11358          0      3          0          6          0
5EE9A4AC 5EE9A5C8  159 TX     327723        374      6          0          6          0
```

此时表上的行级排他锁会阻塞对于表的 DDL 语句：

```
SQL> truncate table scott.emp;
truncate table scott.emp
                     *
ERROR at line 1:
ORA-00054: resource busy and acquire with NOWAIT specified
```

此外，TM 锁定的 ID1 代表的就是锁定的对象号：

```
SQL> select owner,object_name from dba_objects where object_id=11358;
OWNER                          OBJECT_NAME
------------------------------ ------------------------------
SCOTT                          EMP
```

而 TX 锁的 ID1 代表的是事物的回滚段回滚段号、事务槽号，ID2 代表的是顺序号：

```
SQL> select trunc(327723/power(2,16)) ,mod(327723,power(2,16)) from dual;
TRUNC(327723/POWER(2,16)) MOD(327723,POWER(2,16))
------------------------- -----------------------
                        5                      43
```

通过 V$TRANSACTION 视图也可以找到这个事务的信息(注意 XIDSQN 正是 TX 锁的 ID2 信息)：

```
SQL> select xidusn,xidslot,xidsqn from v$transaction;
    XIDUSN    XIDSLOT     XIDSQN
---------- ---------- ----------
         5         43        374
```

如果转储回滚段信息进行分析，再结合 ITL 事务槽，可以清晰地看到锁定的含义以及整个事务的处理过程，也就是上一章中详细探讨过的内容。

2．最常见的锁定：MR 与 AE 锁

可能很多朋友都注意过，在 V$LOCK 视图中，最常见的其实是 MR 锁，也就是介质恢复锁（Media Recovery）：

```
SQL> select * from v$lock where type='MR';
ADDR     KADDR     SID TY        ID1        ID2  LMODE    REQUEST      CTIME      BLOCK
-------- -------- ---- -- ---------- ---------- ------ ---------- ---------- ----------
5FDCFCB8 5FDCFCCC  167 MR          1          0      4          0     196196          0
5FDCFD14 5FDCFD28  167 MR          2          0      4          0     196196          0
5FDCFD70 5FDCFD84  167 MR          3          0      4          0     196196          0
5FDCFDCC 5FDCFDE0  167 MR          4          0      4          0     196196          0
5FDCFE28 5FDCFE3C  167 MR          5          0      4          0     196196          0
5FDCFE84 5FDCFE98  167 MR        201          0      4          0     196196          0
5FDD0164 5FDD0178  167 MR          6          0      4          0      23404          0
5FDD01C0 5FDD01D4  167 MR          8          0      4          0       2892          0


8 rows selected.
```

MR 锁用于保护数据库文件，使得文件在数据库打开、表空间 Online 时不能执行恢复。当进程对数据文件执行恢复时，需要排他的获得 MR 锁。当数据库打开时，每个文件上都分配一个 MR 锁。注意在以上输出中 ID1 代表文件号，其中也包含了 201 号临时文件。

从 Oracle Database 11g 开始，除了每个文件要获得 MR 锁之外，每个登录数据库的会话现在都会缺省获得一个 AE 锁：

```
SQL> select * from v$lock where type='AE' and rownum <5;

ADDR     KADDR           SID TY  ID1  ID2      LMODE    REQUEST      CTIME      BLOCK
-------- -------- ---------- -- ---- ---- ---------- ---------- ---------- ----------
72B670D0 72B670FC         92 AE   99    0          4          0       2317          0
72B671B8 72B671E4         58 AE   99    0          4          0       4817          0
72B6722C 72B67258        157 AE   99    0          4          0   23582614          0
72B67A54 72B67A80         64 AE   99    0          4          0       4816          0
```

现在 MR 锁定和 AE 锁上数据库中最为常见的锁定。

```
SQL> select name from v$event_name where name like '%AE%';
NAME
----------------------------------------------------------------
enq: AE - lock
```

3．ST（空间事务锁）

ST 锁主要用于空间管理和字典管理的表空间(DMT)的区间分配，在 DMT 中典型的是对于 uet$和 fet$数据字典表的争用。对于支持 LMT 的版本，应该尽量使用本地管理表空间，或

者考虑手工预分配一定数量的区（Extent），减少动态扩展时发生的严重队列竞争。以下案例说明了 ST 锁可能会导致的严重性能问题。

```
DB Name         DB Id    Instance     Inst Num Release     OPS Host
------------ ----------- ------------ -------- ----------- --- ------------------
DB          40757346      aaa                1 9.1.7.4.0   NO    server
                Snap Id     Snap Time      Sessions
                ------- ------------------ --------
 Begin Snap:       2845 31-10月-03 02:10:16      46
   End Snap:       2848 31-10月-03 03:40:05      46
    Elapsed:                  89.82 (mins)
```

对于一个 Statspack 的 report，采样时间是非常重要的维度，离开时间做参考，任何等待都不足以说明问题。

```
Top 5 Wait Events
~~~~~~~~~~~~~~~~~                                    Wait          % Total
Event                                                Waits  Time (cs)   Wt Time
-------------------------------------------- ------------ ------------ -------
enqueue                                          53,793   16,192,686   67.86
rdbms ipc message                                 19,999    5,927,350   24.84
pmon timer                                       1,754       538,797    2.26
smon timer                                         17        522,281    2.19
SQL*Net message from client                        94,525    520,104    2.18
          -------------------------------------------------------------
```

在 Statspack 分析中，Top 5 等待事件是我们最为关注的部分。这个系统中，除了 Enqueue 等待事件以外，其他 4 个都属于空闲等待事件，无须关注。来关注一下 Enqueue 等待事件，在 89.82（mins）的采样间隔内，累计 Enqueue 等待长达 16192686cs，即 45 小时左右。这个等待已经太过显著，实际上这个系统也正因此遭遇了巨大的困难，观察到队列等待以后，就应该关注队列等待在等待什么资源。快速跳转的 Statspack 的其他部分，看到以下详细内容：

```
Enqueue activity for DB: DB  Instance: aaa  Snaps: 2716 -2718
-> ordered by waits desc, gets desc

Enqueue            Gets      Waits
---------- ------------ ----------
ST                1,554      1,554
          -------------------------------------------------------------
```

看到主要队列等待在等待 ST 锁定，对于 DMT，我们说这个等待和 FET$、UET$的争用紧密相关。再回过头来研究捕获的 SQL 语句：

```
-> End Buffer Gets Threshold:   10000
-> Note that resources reported for PL/SQL includes the resources used by all SQL statements
   called within the PL/SQL code.  As individual SQL statements are also reported, it is
```

```
   possible and valid for the summed total % to exceed 100
  Buffer Gets    Executions  Gets per Exec  % Total  Hash Value
--------------- ------------ -------------- ------- ------------
      4,800,073       10,268          467.5    51.0   2913840444
select length from fet$ where file#=:1 and block#=:2 and ts#=:3
        803,187       10,223           79.6     9.5    528349613
delete from uet$ where ts#=:1 and segfile#=:2 and segblock#=:3 a
nd ext#=:4
        454,444       10,300           44.1     4.8   1839874543
select file#,block#,length from uet$ where ts#=:1 and segfile#=:
2 and segblock#=:3 and ext#=:4
         23,110       10,230            2.3     0.2   3230982141
insert into fet$ (file#,block#,ts#,length) values (:1,:2,:3,:4)
         21,201          347           61.1     0.2   1705880752
select file# from file$ where ts#=:1
…………
          9,505           12          792.1     0.1   1714733582
select f.file#, f.block#, f.ts#, f.length from fet$ f, ts$ t whe
re t.ts#=f.ts# and t.dflextpct!=0 and t.bitmapped=0
          6,426          235           27.3     0.1   1877781575
delete from fet$ where file#=:1 and block#=:2 and ts#=:3
```

可以看到数据库频繁操作 UET$、FET$系统表已经成为了系统的主要瓶颈。

至此，已经可以准确地为该系统定位问题，相应的解决方案也很容易确定，在 Oracle 9.1.7 中，使用 LMT 代替 DMT，这是解决问题的根本办法，当然实施起来还要进行综合考虑，实际情况还要复杂得多。

8.5.6 Latch Free（闩锁释放）

Latch Free 通常被称为闩锁释放，这个名称常常引起误解，实际上我们应该在前面加上一个“等待”（wait），当数据库出现这个等待时，说明有进程正在等待某个 Latch 被释放，也就是 waiting latch free。

Latch 是一种低级排队（串行）机制，用于保护 SGA 中共享内存结构。Latch 就像是一种快速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。其实不必把 Latch 想得过于复杂，Latch 通常就是操作系统利用内存中的某个区域，通过设置变量为 0 或非 0，来表示 Latch 是否已经被取得，大多数的操作系统，是使用 TEST AND SET 的方式来完成 Latch 检查或持有的。

为了快速的获得一个直观认识，以下示例展现的是 Latch 的示例及获取与释放过程。Latch 在内存中的位置及名称可以通过如下查询获得：

```
SQL> select k.ksmfsadr, ksmfsnam, ksmfstyp, ksmfssiz, kslldnam, kslldlvl
  2  from x$ksmfsv k, x$kslld a
  3  where k.ksmfsnam='ksqeql_' and kslldnam='enqueues';

KSMFSADR KSMFSNAM             KSMFSTYP          KSMFSSIZ KSLLDNAM   KSLLDLVL
-------- -------------------- ---------------- ---------- -------- ----------
20004F60 ksqeql_              ksllt                   100 enqueues          5
```

得到这些信息之后，可以通过 Latch 的地址信息手工对 Latch 进行模拟的持有或释放，注意获取 Latch 使用了 kslgetl 过程，释放 Latch 使用了 kslfre，也就是 Latch Free 过程：

```
SQL> select to_number('20004F60','xxxxxxxxx') from dual;

TO_NUMBER('20004F60','XXXXXXXXX')
---------------------------------
                        536891232

SQL> oradebug setmypid
Statement processed.
SQL> oradebug call kslgetl 536891232 1
Function returned 1
SQL> oradebug call kslfre 536891232
Function returned 0
```

在这个 Latch 的短时持有前后，观察这个 Latch 的等待时间，可以发现大量的 Latch 等待已经发生，这就是 Latch、Latch Get，Latch Free 的一个直观案例：

```
SQL> select name,wait_time from v$latch where name='enqueues';

NAME                                                  WAIT_TIME
-------------------------------------------------- ----------
enqueues                                           2953240015

SQL> /

NAME                                                  WAIT_TIME
-------------------------------------------------- ----------
enqueues                                           3044694486

SQL> select 3044694486 -2953240015 from dual;

3044694486-2953240015
---------------------
             91454471
```

在数据库内部，Oracle 通过 v$latch 视图记录不同类型 Latch 的统计数据，按获取和等待方式不同进行分类，Latch 请求的类型可分为 willing-to-wait 和 immediate 两类。

- willing-to-wait：是指如果所请求的 Latch 不能立即得到，请求进程将等待一段很短的时间后再次发出请求。进程一直重复此过程直到得到 Latch。
- immediate：是指如果所请求的 Latch 不能立即得到，请求进程就不再等待，而是继续执行下去。

在 v$latch 中的以下字段记录了 willing-to-wait 请求。

- GETS：成功地以 willing-to-wait 请求类型请求一个 Latch 的次数。
- MISSES：初始以 willing-to-wait 请求类型请求一个 Latch 不成功，而进程进入等待的次数。
- SLEEPS：初始以 willing-to-wait 请求类型请求一个 Latch 不成功后，进程等待获取 Latch 时进入休眠的次数。

在 v$latch 中的以下字段记录了 immediate 类请求。

- IMMEDIATE_GETS：以 immediate 请求类型成功地获得一个 Latch 的次数。
- IMMEDIATE_MISSES：以 immediate 请求类型请求一个 Latch 不成功的次数。

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次尝试，如果系统存在多个 CPU，那么此进程将围绕该 Latch 开始自旋（spin），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。这一过程可以通过图 9-20 来说明。

旋次数是否达到_spin_count

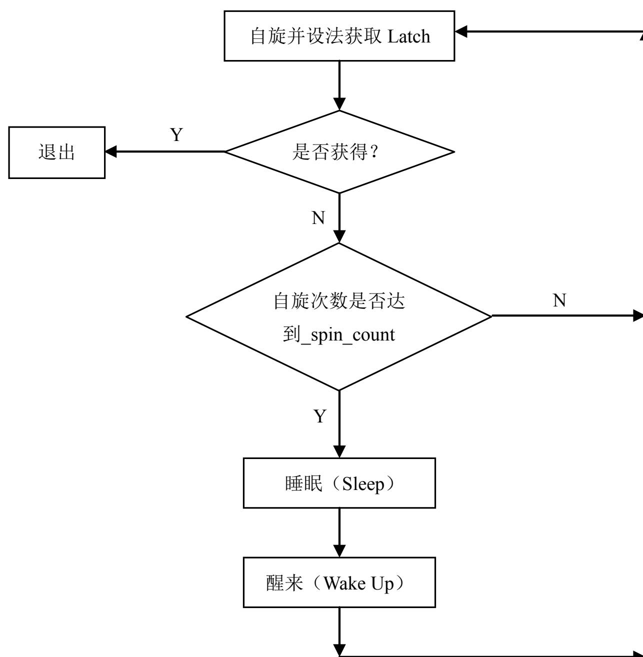


图 9-20　用户进程争夺 Latch 的过程

SPIN 的次数受隐含参数_spin_count 影响，该参数的缺省值为 2000。以下数据取自 Oracle 10gR2 + Linux 环境：

```
SQL> select * from v$version where rownum <2;
BANNER
----------------------------------------------------------------
Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Prod
```

该系统存在 4 颗 CPU：

```
SQL> show parameter cpu_count
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
cpu_count                            integer     4
```

spin_count 的缺省值即为 2000：

```
SQL> @GetHparDes.sql
Enter value for par: spin
old   6:     AND x.ksppinm LIKE '%&par%'
new   6:     AND x.ksppinm LIKE '%spin%'
NAME                 VALUE                DESCRIB
-------------------- -------------------- ------------------------------------------
_spin_count          2000                 Amount to spin waiting for a latch
```

从以上过程可以看到，在 spin 的过程中，进程会一直持有 CPU，spin 的机制是假设 Latch 可以被快速释放（正常情况下 Latch 的持有时间是微秒级，相对 spin 机制如果直接采用 Sleep 方式引起的上下文切换会相当昂贵，所以 Oracle 针对 Latch 引入了 spin 算法），如果其他 CPU 上的其他进程释放了 Latch，SPIN 进程就可以立即获得这个 Latch。如果系统只有单 CPU，那就谈不上 SPIN 了。另一方面也可以看到，Latch 竞争是非常昂贵的，可能导致严重的 CPU 耗用，所以 Latch 竞争在任何时候都应该引起充分的重视。经过 spin 后成功获得 Latch 的次数被记录在 v$latch.spin_gets 字段。下面通过图 9-21 来说明一下 Latch 竞争的情况。

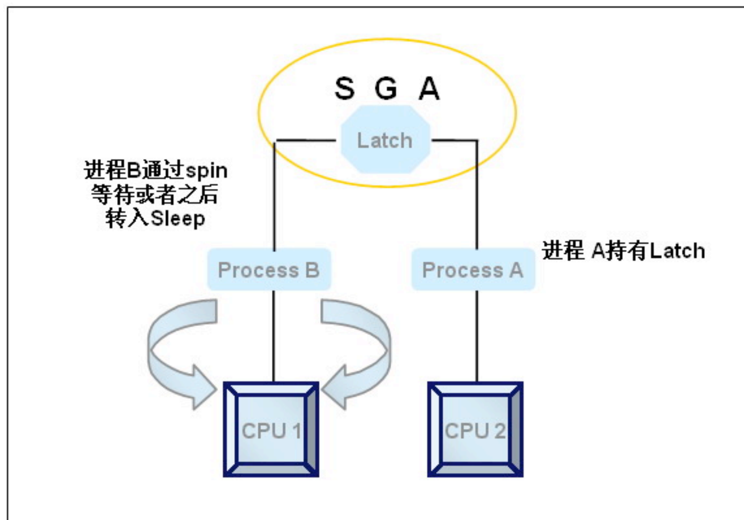


图 9-21　Latch 竞争示意图

继续来具体看一下 willing-to-wait 和 immediate 两类 Latch 的大致数量，以下查询来自 Oracle 10gR2（同以上数据库）：

```
SQL> select count(*) from v$latch;
  COUNT(*)
----------
       382
```

```
SQL> select count(*) from v$latch where IMMEDIATE_GETS + IMMEDIATE_MISSES >0;
  COUNT(*)
----------
        35
SQL> select count(*) from v$latch where IMMEDIATE_GETS + IMMEDIATE_MISSES =0;
  COUNT(*)
----------
       347
```

可以看到 willing-to-wait 类型的等待事件占了绝大部分，immediate 类型的仅为少数：

```
SQL> SELECT   NAME, immediate_gets, immediate_misses, spin_gets
  2      FROM v$latch
  3  WHERE immediate_gets + immediate_misses > 0
  4  ORDER BY immediate_gets DESC;
NAME                                 IMMEDIATE_GETS IMMEDIATE_MISSES  SPIN_GETS
----------------------------------- -------------- ---------------- ----------
cache buffers lru chain                   259891274           209819     213249
cache buffers chains                      258525736             1470      18065
redo copy                                 247810939             7184          0
redo allocation                           247808297             9909        926
checkpoint queue latch                     56443129             4945       3825
simulator lru latch                        18277055              874      12027
cache table scan latch                      7145539             2541          0
SGA IO buffer pool latch                    2468707                0          0
hash table column usage latch                694245                0          0
In memory undo latch                         189592                0       1464
active service list                          181530                0          1
Memory Management Latch                      181372                0          1
SQL memory manager latch                     178333                0          0
KTF sga latch                                176088                0          0
post/wait queue                               66086                0          0
library cache                                 40787                2        104
enqueue hash chains                           26261                1        179
object queue header heap                      11443                0          0
MQL Tracking Latch                            10800                0          0
row cache objects                              5323                0         80
longop free list parent                         553                0          0
process allocation                              479                0          0
msg queue                                        92                0          0
process queue reference                          45                0          0
```

```
kmcpvec latch                                     35          0          0
object stats modification                          3          0          0
query server process                               3          0          0
active checkpoint queue latch                      2          0          1
alert log latch                                    2          0          0
slave class                                        2          0          0
JS mem alloc latch                                 2          0          0
multiblock read objects                            2          0       1160
hash table modification latch                      2          0          0
rules engine evaluation context statistics         2          0          0
RSM SQL latch                                      1          0          0

35 rows selected.
```

需要注意的是，immediate 类型的 Latch 通常是因为存在多个可用 Latch，最常见的如 redo copy latch，当 process 想要取得 redo copy latch 时，它首先要求其中一个 Latch，如果可以取得就持有该 Latch，如果不能获取，它会立刻转向要求另一个 redo copy latch，只有所有的 redo copy latch 都无法取得时，才会 sleep 与 wait。

immediate 的另外一种原因是每个 Latch 都有 level 的概念（level=1-14），当一个 process 需要取得一组 Latches 时，为避免死锁，取得 Latches 有一定的顺序，即 process 新请求的 Latch 的 level，应该大于 process 目前所握有的 Latch 的 level。所以如果 process 要求的新 Latch 的 level 小于目前所持有的 Latch 的 level，正常情况下，Oracle 要求 process 先释放目前所持有的所有 Latch，再依序取得这些 Latch。为节省时间，Oracle 允许进程以 no-wait 方式要求较低 level 的 Latch，如果成功取得，既可以避免 deadlock 又可以节省时间。

在 Oracle 10g 之前，Latch Free 同 Enqueue 一样，是一个汇总等待。从 Oracle 10g 开始，这个等待被分解，现在可以更直接地通过会话等待得知具体的 Latch 发生在哪些资源上：

```
SQL> select name,wait_class
  2  from v$event_name where name like '%latch%';
NAME                                               WAIT_CLASS
-------------------------------------------------- -------------------------
latch: cache buffers chains                        Concurrency
latch: redo writing                                Configuration
latch: redo copy                                   Configuration
latch: Undo Hint Latch                             Concurrency
latch: In memory undo latch                        Concurrency
latch: MQL Tracking Latch                          Concurrency
latch: row cache objects                           Concurrency
latch: shared pool                                 Concurrency
latch: library cache                               Concurrency
latch: library cache lock                          Concurrency
```

```
latch: library cache pin                            Concurrency
……
39 rows selected.
```

最常见的 Latch 集中于 Buffer Cache 的竞争和 Shared Pool 的竞争。和 Buffer Cache 相关的主要 Latch 竞争有 Cache Buffers Chain 和 Cache Buffers LRU Chain，和 Shared Pool 相关的主要 Latch 竞争有 Shared Pool Latch 和 Library Cache Latch 等。

Buffer Cache 的 Latch 竞争经常是由于热点块竞争引起；Shared Pool 的 Latch 竞争通常是由于 SQL 的大量硬解析引起。这些重要的 Latch 竞争曾经在前面章节有过详细论述，此处不再讨论。

在数据库运行环境中，不可避免会出现进程异常终止的情况，而死进程又可能持有 Latch，所以如何释放死进程持有的 Latch 对于数据库来说也是非常重要的。PMON 进程的一个职责就是检测和清除死进程，释放相应的资源，Latch 正是这些资源之一，这一过程通常被称为 Latch Cleanup，共享池（Shared Pool）中有相应的内存结构与此有关：

```
SQL> select * from v$version where rownum <2;
BANNER
----------------------------------------------------------------
Oracle Database 10g Enterprise Edition Release 10.2.0.3.0 - Prod
SQL> select pool,name,bytes from v$sgastat where lower(name) like '%latch cleanup%';
POOL         NAME                                    BYTES
------------ ------------------------------ ----------
shared pool  KTU latch cleanup                         432
shared pool  KTC latch cleanup                         192
```

Latch Cleanup 就是指 PMON 清除死进程持有 Latch 的这一过程，但是要知道 Oracle 要做的工作还远远不止于此，由于 Latch 是用于保护内存结构的关键，如果一个持有 Latch 的进程异常中止，那么相应的内存结构则很可能处于一种不一致的状态，因此 Oracle 需要支持 Latch Recovery。为此，在进程持有 Latch 进行内存结构的修改时，会首先向 Latch Recovery Area 写入相应记录，PMON 进行的工作首先是恢复 Latch 保护的数据结构，然后释放 Latch。

```
SQL> select pool,name,bytes from v$sgastat where lower(name) like '%latch recovery%';

POOL         NAME                                    BYTES
------------ ------------------------------ ----------
shared pool  latch recovery structures                 500
shared pool  latch recovery alignment                   52
```

然而由于 PMON 通常每 3 秒唤醒一次，执行相关的任务，包括 Latch Cleanup，而数据库往往是更加繁忙，所以 Oracle 需要有其他机制来初始化 Latch Cleanup 过程。如果一个进程连续尝试获得 Latch 失败，它将会执行一个 Latch Activity Test 去检查是否需要进行 Latch

Cleanup，如果 Latch 在 5 厘秒之内没有活动，则通知 PMON，PMON 进行检查持有该 Latch 的相关进程是否出现异常以及是否需要进行 Latch 清除工作。

当一个进程执行 Latch activity test 以及等待 PMON 检查持有 Latch 进程的状态时，在数据库中以 latch activity 事件处于等待：

```
SQL> SELECT NAME, parameter1, parameter2, parameter3
  2    FROM v$event_name
  3   WHERE NAME ='latch activity';
NAME                                               PARAMETER1 PARAMETER2 PARAMETER3
-------------------------------------------------- ---------- ---------- ----------
latch activity                                     address    number     process#
```

Latch activity 的第一个参数是 Latch 的内存地址，参数 2 代表 Latch 的类型，参数 3 代表持有 Latch 的进程号（0 代表正在进行 Latch Activity Test）

有时候在数据库关闭时（甚至是 abort 关闭时），可能会看到如下提示 PMON failed to acquire latch，这就是指在关闭数据库时，PMON 进程不能及时终止进程，释放相关的内存锁定：

```
Tue Jul 25 08:33:21 2006
Shutting down instance (abort)
License high water mark = 639
Tue Jul 25 08:33:21 2006
PMON failed to acquire latch, see PMON dump
PMON failed to acquire latch, see PMON dump
PMON failed to acquire latch, see PMON dump
PMON failed to acquire latch, see PMON dump
PMON failed to acquire latch, see PMON dump
PMON failed to acquire latch, see PMON dump
PMON failed to acquire latch, see PMON dump
PMON failed to acquire latch, see PMON dump
PMON failed to acquire latch, see PMON dump
```

在这种情况下，通常通过手工杀掉操作系统进程等方式可以加快数据库的关闭。

8.5.7 Oracle 10g/11g Latch 机制的变化

前面曾经提到，Oracle 的 Latch 机制采用 spin 来进行持有 CPU 的不断尝试，虽然通常 Latch 的获取会非常快（一般在微秒级），但是很多时候 Latch 竞争还是为引发极为严重的 CPU 争用。所以从 Oracle 10g 开始，Oracle 尝试引入了一种新的机制来代替传统的 Latch 机制，这就是 Mutex 机制，也就是互斥机制。和 Latch 相比，一个 Mutex Get 大约仅需要 30～35 个指令，而 Latch Get 则需要大约 150～200 个指令，同时在大小上，每个 Mutex 仅占用大约 16 Bytes 空间，而 Latch 在 10gR2 中要占用大约 112 Bytes 空间。

在 Oracle 10.2.0.1 中一个新的参数_kks_use_mutex_pin 被引入，不过缺省值为 False：

```
SQL> SELECT x.ksppinm NAME, y.ksppstvl VALUE, x.ksppdesc describ
  2    FROM SYS.x$ksppi x, SYS.x$ksppcv y
  3   WHERE x.indx = y.indx
  4     AND x.ksppinm LIKE '%&par%'
  5  /
Enter value for par: kks
old   4:    AND x.ksppinm LIKE '%&par%'
new   4:    AND x.ksppinm LIKE '%kks%'

NAME                           VALUE  DESCRIB
------------------------------ ------ ------------------------------
_kks_use_mutex_pin             FALSE  Turning on this will make KKS use mutex for cursor pins.
```

这意味着新的 Mutex 机制已经准备好了用于应付 Cursor Pin 处理。在 Oracle 10gR2 随后的版本中，这个参数被设置为 True：

```
SQL> select * from v$version where rownum <2;
BANNER
----------------------------------------------------------------
Oracle Database 10g Enterprise Edition Release 10.2.0.3.0 - 64bi
SQL> @GetHidPar
Enter value for par: mutex
NAME                           VALUE  DESCRIB
------------------------------ ------ ------------------------------------------------
_kks_use_mutex_pin             TRUE   Turning on this will make KKS use mutex for cursor pins.
```

Mutex 机制首先被引入用于替代 Library Cache Latch 以及 Library Cache Pin 等机制。使用 Mutex Pin 机制 Oracle 能够使用更少的 CPU 资源获得更好的性能。由于 Mutex Pin 机制带来的 Cursor 管理上的性能提升，所以曾经用来缓解 Cursor Latch 竞争的参数 CURSOR_SPACE_FOR_TIME 从 Oracle 10.2.0.5 和 Oracle 11.1.0.7 开始将不再被支持。

在 Oracle 10.2.0.3 中，以下是数据库中和 Library Cache 相关的 Latch 与等待：

```
SQL> select * from v$version where rownum <2;
BANNER
----------------------------------------------------------------
Oracle Database 10g Enterprise Edition Release 10.2.0.3.0 - 64bi
SQL> select name from v$latch where name like '%library%';
NAME
------------------------------
library cache
library cache pin allocation
library cache lock allocation
```

```
library cache pin
library cache load lock
library cache lock
library cache hash chains
SQL> select name from v$event_name where name like '%library%';
NAME
------------------------------
latch: library cache
latch: library cache lock
latch: library cache pin
library cache pin
library cache lock
library cache load lock
library cache revalidation
library cache shutdown
```

在 Oracle Database 11g 中，和 Mutex 相关的数据库描述已经被引入，首先的变化是和 Library Cache 相关的 Latch 仅余 Library Cache Load Lock：

```
SQL> select * from v$version where rownum <2;
BANNER
--------------------------------------------------------------------------------
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
SQL> select name from v$latch where name like '%library%';
NAME
----------------------------------------------------------------
library cache load lock
```

其次是等待事件引入了 Library Cache:Mutex S/X 等待，用以描述因为 Mutex 竞争而导致的 Library Cache 等待：

```
SQL> select name from v$event_name where name like '%library%';
NAME
----------------------------------------------------------------
library cache pin
library cache lock
library cache load lock
library cache: mutex X
library cache: mutex S
OSD IPC library
library cache revalidation
library cache shutdown
```

虽然在 Oracle Database 11gR1 中，这些变化已经可以清晰地看到，但是一些问题仍然存

在，Oracle 仍然在通过一系列不断努力和改进使得这些新的变化走向成熟。

当然 Library Cache Latch 在某些条件下仍然会被用到，Oracle 数据库的很多内容还在不断变化中。关于 Mutex 的信息主要可以通过两个视图来查询：v$mutex_sleep 和 v$mutex_sleep_history。以下是一个 Oracle 11g 生产环境中的查询示范输出：

```
SQL> select SLEEP_TIMESTAMP,MUTEX_TYPE,GETS,SLEEPS,LOCATION,MUTEX_VALUE
  2  from v$mutex_sleep_history where rownum <10;
SLEEP_TIMESTAMP               MUTEX_TYPE        GETS SLEEPS LOCATION             MUTEX_VA
----------------------------- ------------- -------- ------ -------------------- --------
10-OCT-08 10.51.13.286692 PM  Library Cache  2396354   2888 kgllkc1   57         00730000
27-SEP-08 06.26.06.656187 AM  Library Cache  2481002   2082 kglhdgn1  62         00660000
27-JUL-08 09.14.54.486931 AM  Library Cache  1923536   1009 kglpnal1  90         00
06-OCT-08 07.09.10.242297 PM  Library Cache  2566850   1038 kglpnal1  90         00
06-SEP-08 06.06.43.826838 AM  Library Cache  2291540   1953 kglget2   2          00
24-JUL-08 01.29.49.457944 PM  Cursor Pin     1781869    834 kkslce [KKSCHLPIN2]  003E0000
01-AUG-08 09.39.30.433336 PM  Library Cache  1967534   1928 kglget2   2          00
14-SEP-08 06.09.49.020290 AM  Library Cache   588263     22 kglhdgn1  62         00280000
22-APR-08 07.46.32.370000 PM  Library Cache  1366596     31 kglpnal1  90         00
```

在 Oracle 10g 中，如果注意一下，在一些进程转储或者跟踪文件中已经可以看到 Mutex 的相关信息：

```
bash-2.05$ grep -i mutex smsdbn2_ora_889.trc
       Mutex 0(0, 0) idn 0 oper NONE
       Mutex 0(0, 0) idn 0 oper NONE
       Mutex 0(0, 0) idn 0 oper NONE
       Mutex 3c57e4d88(0, 1) idn 32e8da0b oper SHRD
word kksmutexpin_ [105F0D004, 105F0D008) = 00000317
boolean KKSUSEMUTEXPIN_ [380018C60, 380018C64) = 00000001
```

Oracle 数据库的很多内部变化发生在潜移默化之中，这里提到的 Latch 变化只不过是冰山一角。只有不断进行研究与学习，才有可能跟上 Oracle 变化的步伐。

Oracle 的等待事件不可能一一列举，本章提供的一些思路和方法，希望可以使得大家对于 Oracle 的等待事件有一个初步的认识，并掌握一些研究和深入的方法。

参考信息及建议阅读

（1）Oracle® Database Reference 11g Release 1 (11.1)　　Part Number B28320-01
（2）CURSOR_SPACE_FOR_TIME To Be Deprecated　　Metalink Note:565424.1
（3）一个命题:列举你认为最重要的 9 个动态性能视图
http://www.eygle.com/archives/2005/12/9_most_important_views.html

（4）列举你认为最重要的 9 个动态性能视图

http://www.itpub.net/471751.html

（5）My answer for-9 个动态性能视图

http://www.eygle.com/archives/2005/12/my_answer_for_9_views.html