

## Statspack 使用指南 ( Eyle [evgle@itpub.net](mailto:evgle@itpub.net) )

### 作者介绍：

盖国强，曾任ITPUB MS版版主，现任Oracle数据库管理版版主。曾任职于某国家大型企业，服务于烟草行业，开发过基于Oracle数据库的大型ERP系统，属国家信息产业部重点工程。同时负责Oracle数据库管理及优化，并为多家烟草企业提供Oracle数据库管理、优化及技术支持。目前任职于北京某电信增值业务系统提供商企业，负责数据库业务。管理全国30多个省点数据库平台。实践经验丰富，长于数据库诊断、优化与SQL调整。希望与大家共同学习提高Oracle技术水平。

Oracle Statspack 从 Oracle8.1.6 开始被引入 Oracle,并马上成为 DBA 和 Oracle 专家用来诊断数据库性能的强大有力的工具。通过 Statspack 我们可以很容易的确定 Oracle 数据库的瓶颈所在，记录数据库性能状态，也可以使远程技术支持人员迅速了解你的数据库运行状况。因此了解和使用 Statspack 对于 DBA 来说至关重要。

在数据库中 Statspack 的脚本位于\$ORACLE\_HOME/RDBMS/ADMIN 目录下，对于 ORACLE8.1.6,是一组以 stat 开头的文件；对于 ORACLE8.1.7,是一组以 sp 开头的文件。

在 Oracle8.1.6 中，Statspack 第一次发布，但是你也可以在以下链接找到可用于 Oracle8.0~Oracle8.1.5 的脚本。

<http://www.oracle.com/oramag/oracle/00-Mar/index.html?o20tun.html>  
<http://www.oracle.com/oramag/oracle/00-Mar/index.html?statspack-other.html>  
<http://www.oracle.com/oramag/oracle/00-Mar/index.html?statspack.tar>

在 816 以前的版本使用 Statspack，你需要使用 statscbps.sql 脚本建立一个 v\$buffer\_pool\_statistics 视图，该脚本包含在以上链接下载的 tar 文件中。

访问该链接，你可能需要一个 OTN 帐号，申请该帐号是免费的。

在 Statspack 发布之前，我们通常能够使用诊断数据库的工具是两个脚本 UTLBSTAT.SQL 和 UTLESTAT.SQL，BSTAT/ESTAT 是一个非常简单的性能诊断工具。UTLBSTAT 获得开始时很多 V\$视图的快照，UTLESTAT 通过先前的快照和当前视图生成一个报表。

该报表实际上相当于 statspack 中的两个采样点。

Statspack 通过连续的采样，能够给我们提供至关重要的趋势分析数据。这是一个巨大的进步。

能够使用 Statspack 的环境我们就尽量不要使用 BSTAT/ESTAT 的方式来诊断数据库问题。

下面我们来讲一讲 Statspack 的安装，配置，使用 and 解读，通过这篇文章，我们希望至少可以使每个使用 Oracle 数据库的人，都可以学会怎样生成 Statspack Report.

## 一. 系统参数

为了能够顺利安装和运行 Statspack 你可能需要设置以下系统参数：

### 1. job\_queue\_processes

为了能够建立自动任务，执行数据收集，该参数需要大于 0。你可以在初始化参数文件中修改该参数(使该参数在重启后依然有效)。该参数可以在系统级动态修改(重启后失效)。

```
SQL> alter system set job_queue_processes = 6;
System altered
```

在 Oracle9i 当中，可以指定范围，如 both,这样该修改在当前及之后保持有效(仅当你使用 spfile 时，如果在 9i 中仍然使用 pfile，那么更改方法同 8i 相同)：

```
SQL> alter system set job_queue_processes = 6 scope=both;
系统已更改。
```

### 2. timed\_statistics

收集操作系统的计时信息，这些信息可被用来显示时间等统计信息、优化数据库和 SQL 语句。要防止因从操作系统请求时间而引起的开销，请将该值设置为 False。

使用 statspack 收集统计信息时建议将该值设置为 TRUE，否则收集的统计信息大约只能起到 10%的作用，将 timed\_statistics 设置为 True 所带来的性能影响与好处相比是微不足道的。

该参数使收集的时间信息存储在在 V\$SESSTATS 和 V\$SYSSTATS 等动态性能视图中。

Timed\_statistics 参数可以在实例级进行更改

```
SQL> alter system set timed_statistics = true;
System altered
SQL>
```

如果你担心一致启用 timed\_statistics 对于性能的影响，你可以在使用 statspack 之前，在 system 更改，采样过后把该参数动态修改成 false。

## 二. 安装 Statspack

安装 Statspack 需要用 internal 身份登陆, 或者拥有 SYSDBA(connect / as sysdba)权限的用户登陆。需要在本地安装或者通过 telnet 登陆到服务器。

在 Oracle8.1.6 版本中运行 statscre.sql;在 Oracle8.1.7 版本中运行 spcreate.sql。

首先登陆到数据库, 最好转到\$ORACLE\_HOME/RDBMS/ADMIN 目录, 这样我们执行脚本就可以方便些。

```
D:\oracle\ora81\RDBMS\ADMIN>sqlplus internal
SQL*Plus: Release 8.1.7.0.0 - Production on 星期二 12月 3 16:54:53 2002
(c) Copyright 2000 Oracle Corporation. All rights reserved.

请输入口令:

连接到:
Oracle8i Enterprise Edition Release 8.1.7.0.0 - Production
With the Partitioning option
JServer Release 8.1.7.0.0 - Production

SQL> select instance_name,host_name,version,startup_time from v$instance;

INSTANCE_NAME  HOST_NAME  VERSION          STARTUP_TIME
-----
eygle          AM-SERVER  8.1.7.0.0        22-11月-02

SQL>
```

**注:在 Oracle9i 中, 不存在 internal 用户, 可以使用 sys 用户以 sysdba 身份连接:**

```
D:\oracle\ora92\rdbms\admin>sqlplus "/ as sysdba"
SQL*Plus: Release 9.2.0.3.0 - Production on 星期四 7月 10 19:18:54 2003
Copyright (c) 1982, 2002, Oracle Corporation. All rights reserved.

连接到: Oracle9i Enterprise Edition Release 9.2.0.3.0 - Production
With the Partitioning, Oracle Label Security, OLAP and Oracle Data Mining options
JServer Release 9.2.0.3.0 - Production

SQL>
```

检查数据文件路径及磁盘空间, 以决定创建数据文件的位置:

```
SQL> select file_name from dba_data_files;
FILE_NAME
-----
D:\ORACLE\ORADATA\EYGLE\SYSTEM01.DBF
D:\ORACLE\ORADATA\EYGLE\TEMP01.DBF
.....
D:\ORACLE\ORADATA\EYGLE\HH_AM01.ORA

已选择24行。

SQL>
```

创建存储数据的表空间，如果采样间隔较短，周期较长，打算长期使用，那么你可能需要一个大一点的表空间，如果每个半个小时采样一次，连续采样一周，数据量是很大的。本例创建一个 500M 的测试表空间。

**注意:这里创建的表空间不能太小，如果太小创建对象会失败，至少需要建立 100M 表空间,如果打算长期使用,可以建立稍大的表空间，本例创建 500M LMT 表空间。**

```
SQL> create tablespace perfstat
 2  datafile 'd:\oracle\oradata\eygle\perfstat.dbf'
 3  size 500M
 4  extent management local;
```

表空间已创建。

SQL>

检查是否存在安装所需要的脚本文件(对于不同的版本，脚本有所不同)

```
E:\Oracle\ora92\rdbms\admin>dir /w sp*
```

驱动器 E 中的卷没有标签。

卷的序列号是 ACC3-4340

E:\Oracle\ora92\rdbms\admin 的目录

|              |             |                    |             |              |              |
|--------------|-------------|--------------------|-------------|--------------|--------------|
| spauto.sql   | spcpkg.sql  | screate.sql        | spctab.sql  | spcusr.sql   | spdoc.txt    |
| spdrop.sql   | spdtab.sql  | spdusr.sql         | sppurge.sql | sprepins.sql | spreport.sql |
| sprepsql.sql | sptrunc.sql | spuexp.par         | spup816.sql | spup817.sql  | spup90.sql   |
|              | 18 个文件      | 510,296 字节         |             |              |              |
|              | 0 个目录       | 4,146,565,120 可用字节 |             |              |              |

接下来我们就可以开始安装 Statspack 了。这期间会提示你输入缺省表空间和临时表空间的位置,输入我们为 perfstat 用户创建的表空间和你的临时表空间。

```
SQL> @screate
```

```
.
.
Specify PERFSTAT user's default tablespace
输入 default_tablespace 的值: perfstat
Using perfstat for the default tablespace
```

用户已更改。

用户已更改。

```
Specify PERFSTAT user's temporary tablespace
输入 temporary_tablespace 的值: temp
```

**注意:在 statspack 创建过程中,当提示输入口令时，你可以输入一个明文口令，但是如果输入口令不符合规范(如 123 或以数字开头的口令)，创建会失败。**

输入口令时可以暂时输入:perfstat ,稍后可以更改。

```
... Creating PERFSTAT user ...
```

```
Choose the PERFSTAT user's password.
```

```
Not specifying a password will result in the installation FAILING
```

```
Specify PERFSTAT password
```

```
输入 perfstat_password 的值: 123
```

123

PL/SQL 过程已成功完成。

create user perfstat identified by 123

\*

ERROR 位于第 1 行:

ORA-00988: 缺少或无效口令

如果安装成功，你可以看到如下的输出信息：

....

Creating Package STATSPACK...

程序包已创建。

没有错误。

Creating Package Body STATSPACK...

程序包主体已创建。

没有错误。

NOTE:

SPCPKG complete. Please check spcpkg.lis for any errors.

你可以查看.lis 文件查看安装时的错误信息。

SQL> host dir \*.lis

驱动器 D 中的卷没有标签。

卷的序列号是 5070-5982

D:\oracle\ora81\RDBMS\ADMIN 的目录

2002-12-03 17:25 204 spcpkg.lis

2002-12-03 17:25 2,276 spctab.lis

2002-12-03 17:25 3,965 spcusr.lis

2002-12-03 17:23 1,187 spdtab.lis

2002-12-03 17:24 351 spdusr.lis

5 个文件 7,983 字节

0 个目录 3,965,304,832 可用字节

SQL> host find "ORA-" \*.lis

SQL> host find "err" \*.lis

----- SPAUTO.LIS

----- SPCPKG.LIS

SPCPKG complete. Please check spcpkg.lis for any errors.

----- SPCTAB.LIS

SPCTAB complete. Please check spctab.lis for any errors.

----- SPCUSR.LIS

SPCUSR complete. Please check spcusr.lis for any errors.

----- SPDTAB.LIS

在 UNIX 上，你可以通过以下命令查看相应的错误信息

```
$ ls *.lis
spauto.lis  spcpkg.lis  spctab.lis  spcusr.lis  spdtab.lis  spdusr.lis
$ grep ORA- *.lis
$ grep err *.lis
spcpkg.lis:SPCPKG complete. Please check spcpkg.lis for any errors.
spctab.lis:SPCTAB complete. Please check spctab.lis for any errors.
spcusr.lis:SPCUSR complete. Please check spcusr.lis for any errors.
spdtab.lis:SPDTAB complete. Please check spdtab.lis for any errors.
spdusr.lis:SPDUSR complete. Please check spdusr.lis for any errors.
```

在这一步，如果出现错误，那么你可以运行 `spdrop.sql` 脚本来删除这些对象。然后重新运行 `spcreate.sql` 来创建这些对象。运行 SQL\*Plus，以具有 SYSDBA 权限的用户登陆：

```
SQL> @spdrop.sql
.
.
.
同义词已丢弃。 off;

视图已丢掉。

同义词已丢弃。

视图已丢掉。

同义词已丢弃。

用户已丢弃

NOTE:
SPDUSR complete. Please check spdusr.lis for any errors.

SQL>
```

### 三. 测试安装好的 Statspack

运行 statspack.snap 可以产生系统快照，运行两次，然后执行 spreport.sql 就可以生成一个基于两个时间点的报告。

如果一切正常，说明安装成功。

```
SQL>execute statspack.snap
PL/SQL procedure successfully completed.
SQL>execute statspack.snap
PL/SQL procedure successfully completed.
SQL>@spreport.sql
...
```

可是有可能你会得到以下错误：

```
SQL> exec statspack.snap;
BEGIN statspack.snap; END;

*
ERROR at line 1:
ORA-01401: inserted value too large for column
ORA-06512: at "PERFSTAT.STATSPACK", line 978
ORA-06512: at "PERFSTAT.STATSPACK", line 1612
ORA-06512: at "PERFSTAT.STATSPACK", line 71
ORA-06512: at line 1
```

这是 Oracle 的一个 Bug，Bug 号 **1940915**。

该 Bug 自 8.1.7.3 后修正。

这个问题只会出现在多位的字符集,需要修改 spcpkg.sql 脚本，\$ORACLE\_HOME/rdbms/admin/spcpkg.sql，将"substr" 修改为 "substrb"，然后重新运行该脚本。

该脚本错误部分：

```
select l_snap_id
, p_dbid
, p_instance_number
, substr(sql_text,1,31)
.....
```

substr 会将多位的字符，当作一个 byte.substrb 则会当作多个 byte。在收集数据时，statspack 会将 top 10 的 sql 前 31 个字节 存入数据表中,若在 SQL 的前 31 个字有中文，就会出现此错误。

#### 四. 规划自动任务

Statspack 正确安装以后，我们就可以设置定时任务，开始收集数据了。可以使用 spauto.sql 来定义自动任务。

先来看看 spauto.sql 的关键内容：

```
dbms_job.submit(:jobno, 'statspack.snap',  
trunc(sysdate+1/24,'HH'), 'trunc(SYSDATE+1/24,"HH")', TRUE, :instno);
```

这个 job 任务定义了收集数据的时间间隔：

一天有 24 个小时，1440 分钟，那么：

|         |        |
|---------|--------|
| 1/24 HH | 每小时一次  |
| 1/48 MI | 每半小时一次 |
| 1/144MI | 每十分钟一次 |
| 1/288MI | 每五分钟一次 |

我们可以修改 spauto.sql 来更改执行间隔，如：

```
dbms_job.submit(:jobno, 'statspack.snap',  
trunc(sysdate+1/48,'MI'), 'trunc(SYSDATE+1/48,"MI")', TRUE, :instno);
```

然后我们执行 spauto，这样我们就建立了一个每 30 分钟执行一次的数据收集计划。你可以查看 spauto.lis 来获得输出信息：

```
SQL> @spauto  
PL/SQL procedure successfully completed.  
  
Job number for automated statistics collection for this instance  
~~~~~  
Note that this job number is needed when modifying or removing  
the job:  
  
      JOBNO  
-----  
        28  
  
Job queue process  
~~~~~  
Below is the current setting of the job_queue_processes init.ora  
parameter - the value for this parameter must be greater  
than 0 to use automatic statistics gathering:  
  
NAME                                TYPE                                VALUE  
-----  
job_queue_processes                 integer                             5  
  
Next scheduled run  
~~~~~  
The next scheduled run for this job is:
```

| JOB | NEXT_DATE | NEXT_SEC |
|-----|-----------|----------|
| 28  | 15-AUG-03 | 16:00:00 |

关于采样间隔，我们通常建议以 1 小时为时间间隔，对于有特殊需要的环境，可以设置更短的，如半小时作为采样间隔，但是不推荐更短。因为 statspack 的执行本身需要消耗资源，对于繁忙的生产系统，太短的采样对系统的性能会产生较大的影响（甚至会使 statspack 的执行出现在采样数据中）。

## 五. 生成分析报告

调用 spreport.sql 可以生成分析报告：

```
SQL> @spreport

  DB Id      DB Name      Inst Num Instance
-----
1277924236 EYGLE                1 eygle

Completed Snapshots

Instance      DB Name      Snap Id      Snap Started      Snap Level Comment
-----
eygle        EYGLE        1 04 12 月 2002 14:4      5
              8
              2 04 12 月 2002 15:0      5
              0
.....
              98 05 12 月 2002 04:1      5
              3
eygle        EYGLE        99 05 12 月 2002 04:2      5
              3
              100 05 12 月 2002 04:3      5
              3

Specify the Begin and End Snapshot Ids
~~~~~
输入 begin_snap 的值: 1
Begin Snapshot Id specified: 1

输入 end_snap 的值: 100
End Snapshot Id specified: 100

Specify the Report Name
~~~~~
The default report file name is sp_1_100. To use this name,
press <return> to continue, otherwise enter an alternative.
输入 report_name 的值: rep1205.txt

Using the report name rep1205.txt
```

这样就生成了一个报告，可是如果中间停过机，那么你可能收到以下错误信息：

```
ERROR: Snapshots chosen span an instance shutdown: RESULTS ARE INVALID
STATSPACK report for

DB Name      DB Id      Instance      Inst Num Release      OPS Host
-----
EYGLE        1277924236 eygle                1 8.1.7.0.0      NO AM-SERVER
```

```
.ela      := ;
          *
ERROR 位于第 4 行:
ORA-06550: 第 4 行, 第 17 列:
PLS-00103: 出现符号 ";"在需要下列之一时 :
(-+modnotnull<an identifier>
<a double-quoted delimited-identifier><a bind variable>avg
countcurrentexistsmaxminpriorsqlstddevsumvarianceexecute
foralltimeintervaldate
<a string literal with character set specification>
<a number><a single-quoted SQL string>
符号 "null" 被替换为 ";" 后继续。
ORA-06550: 第 6 行, 第 16 列:
PLS-00103: 出现符号 ";"在需要下列之一时 :
(-+modnotnull<an identifier>
<a double-quoted delimited-identifier><a bind variable>avg
countcurrentexistsmaxminpriorsqlstddevsumvarianceexecute
foralltimeintervaldate
<a stri
```

一个 statspack 的报告不能跨越一次停机，但是之前或之后的连续区间，收集的信息依然有效。你可以选择之前或之后的采样声称 report。

## 六. 移除定时任务

移除一个定时任务，可以如下操作:

```
SQL> select job,log_user,priv_user,last_date,next_date,interval from user_jobs;
JOB LOG_USER    LAST_DATE      NEXT_DATE      INTERVAL
-----
22 PERFSTAT     2002-12-5:14:33:26 2002-12-5 14:43:00 trunc(SYSDATE+1/144,'MI')

SQL> execute dbms_job.remove('22')

PL/SQL procedure successfully completed
```

当你完成了一个采样报告，你应该及时移除这个 job 任务，在生产环境中，遗漏一个无人照顾的 job 是非常危险的，如果 statspack 运行一个星期，采样的数据量是非常惊人的。有的生产企业因疏忽而当机！

## 七. 删除历史数据

删除 stats\$snapshot 数据表中的相应数据，其他表中的数据会相应的级连删除：

```
SQL> select max(snap_id) from stats$snapshot;

MAX(SNAP_ID)
-----
          166

SQL> delete from stats$snapshot where snap_id <= 166;

143 rows deleted
```

你可以更改 snap\_id 的范围以保留你需要的数据。  
在以上删除过程中，你可以看到所有相关的表都被锁定。

```
SQL> select a.object_id,a.oracle_username ,b.object_name
       from v$locked_object a,dba_objects b
       where a.object_id = b.object_id
       /

OBJECT_ID ORACLE_USERNAME          OBJECT_NAME
-----
156 PERFSTAT                      SNAP$
39700 PERFSTAT                     STAT$LIBRARYCACHE
39706 PERFSTAT                     STAT$ROLLSTAT
39712 PERFSTAT                     STAT$SGA
39754 PERFSTAT                     STAT$PARAMETER
39745 PERFSTAT                     STAT$SQL_STATISTICS
39739 PERFSTAT                     STAT$SQL_SUMMARY
39736 PERFSTAT                     STAT$ENQUEUESTAT
39733 PERFSTAT                     STAT$WAITSTAT
39730 PERFSTAT                     STAT$BG_EVENT_SUMMARY
39724 PERFSTAT                     STAT$SYSTEM_EVENT
39718 PERFSTAT                     STAT$SYSSTAT
39715 PERFSTAT                     STAT$SGASTAT
39709 PERFSTAT                     STAT$ROWCACHE_SUMMARY
39703 PERFSTAT                     STAT$BUFFER_POOL_STATISTICS
39697 PERFSTAT                     STAT$LATCH_MISSES_SUMMARY
39679 PERFSTAT                     STAT$SNAPSHOT
39682 PERFSTAT                     STAT$FILESTATXS
39688 PERFSTAT                     STAT$LATCH
174 PERFSTAT                       JOBS

20 rows selected
```

Oracle 还提供了系统脚本用于 Truncate 这些统计信息表，这个脚本名字是: sptrunc.sql (8i、9i 都相同)  
该脚本主要内容如下，里面看到的就是 statspack 相关的所有系统表：

```
truncate table STAT$FILESTATXS;
truncate table STAT$LATCH;
truncate table STAT$LATCH_CHILDREN;
truncate table STAT$LATCH_MISSES_SUMMARY;
truncate table STAT$LATCH_PARENT;
```

```
truncate table STATS$LIBRARYCACHE;
truncate table STATS$BUFFER_POOL_STATISTICS;
truncate table STATS$ROLLSTAT;
truncate table STATS$ROWCACHE_SUMMARY;
truncate table STATS$SGA;
truncate table STATS$SGASTAT;
truncate table STATS$SYSSTAT;
truncate table STATS$SESSTAT;
truncate table STATS$SYSTEM_EVENT;
truncate table STATS$SESSION_EVENT;
truncate table STATS$BG_EVENT_SUMMARY;
truncate table STATS$WAITSTAT;
truncate table STATS$ENQUEUESTAT;
truncate table STATS$SQL_SUMMARY;
truncate table STATS$SQL_STATISTICS;
truncate table STATS$SQLTEXT;
truncate table STATS$PARAMETER;

delete from STATS$SNAPSHOT;
delete from STATS$DATABASE_INSTANCE;

commit;
```

如果采样了大量的数据，直接 Delete 是非常缓慢的，可以考虑使用上述 SQL 截断所有表。

## 八. 其它重要脚本

### 1. 通过导出保存及共享数据

在诊断系统问题时,可能需要向专业人士提供原始数据,这时我们可以导出 Statspack 表数据,其中我们可能用到:spuexp.par  
其内容主要为:

```
file=spuexp.dmp log=spuexp.log compress=y grants=y indexes=y rows=y constraints=y  
owner=PERFSTAT consistent=y
```

我们可以导出如下:

```
exp userid=perfstat/my perfstat password parfile=spuexp.par
```

### 2. 删除数据

spdrop.sql 在执行时主要调用两个脚本: spdtab.sql、spdusr.sql  
前者删除表及同义词等数据,后者删除用户

### 3. Oracle92 中新增加的脚本

1. 用于升级 statspack 对象的脚本,这些脚本需要以具有 SYSDBA 权限的用户运行,升级前请先备份存在的 Schema 数据:

SPUP90.SQL: 用于升级 9.0 版本的模式至 9.2 版本。

SPUP817.SQL: 如果从 Statspack 8.1.7 升级,需要运行这个脚本

SPUP816.SQL: 从 Statspack 8.1.6 升级,需要运行这个脚本,然后运行 SPUP817.SQL.

2. sprepsql.sql 用于根据给定的 SQL Hash 值生成 SQL 报告

## 九. 调整 STATSPACK 的收集门限

Statspack 有两种类型的收集选项：

级别 ( level ) : 控制收集数据的类型

门限 ( threshold ) : 设置收集的数据的阈值。

### 1. 级别 ( level )

Statspack 共有三种快照级别，默认值是 5

a.level 0: 一般性能统计。包括等待事件、系统事件、系统统计、回滚段统计、行缓存、SGA、会话、锁、缓冲池统计等等。

b.level 5: 增加 SQL 语句。除了包括 level0 的所有内容，还包括 SQL 语句的收集，收集结果记录在 stats\$sql\_summary 中。

c.level 10: 增加子锁存统计。包括 level5 的所有内容。并且还会将附加的子锁存存入 stats\$latch\_children 中。在使用这个级别时需要慎重，建议在 Oracle support 的指导下进行。

可以通过 statspack 包修改缺省的级别设置

```
SQL>execute statspack.snap(i_snap_level=>0,i_modify_parameter=>'true');
```

通过这样的设置，以后的收集级别都将是 0 级。

如果你只是想本次改变收集级别，可以忽略 i\_modify\_parameter 参数。

```
SQL>execute statspack.snap(i_snap_level=>10);
```

### 2. 快照门限

快照门限只应用于 stats\$sql\_summary 表中获取的 SQL 语句。

因为每一个快照都会收集很多数据，每一行都代表获取快照时数据库中的一个 SQL 语句，所以 stats\$sql\_summary 很快就会成为 Statspack 中最大的表。

门限存储在 stats\$statspack\_parameter 表中。让我们了结一下各种门限：

a. executions\_th 这是 SQL 语句执行的数量(默认值是 100)

b. disk\_reads\_th 这是 SQL 语句执行的磁盘读入数量 (默认值是 1000)

c. parse\_calls\_th 这是 SQL 语句执行的解析调用的数量 (默认值是 1000)

d. buffer\_gets\_th 这是 SQL 语句执行的缓冲区获取的数量 (默认值是 10000)

任何一个门限值超过以上参数就会产生一条记录。

通过调用 statspack.modify\_statspack\_parameter 函数我们可以改变门限的默认值。

例如：

```
SQL>execute statspack.modify_statspack_parameter(i_buffer_gets_th=>100000,i_disk_reads_th=>100000);
```

## 一〇.整理分析结果

可以通过各种工具建立图表,使我们收集的数据更直观,更有说服力。  
以下是我给一个客户做的分析报告的实例。

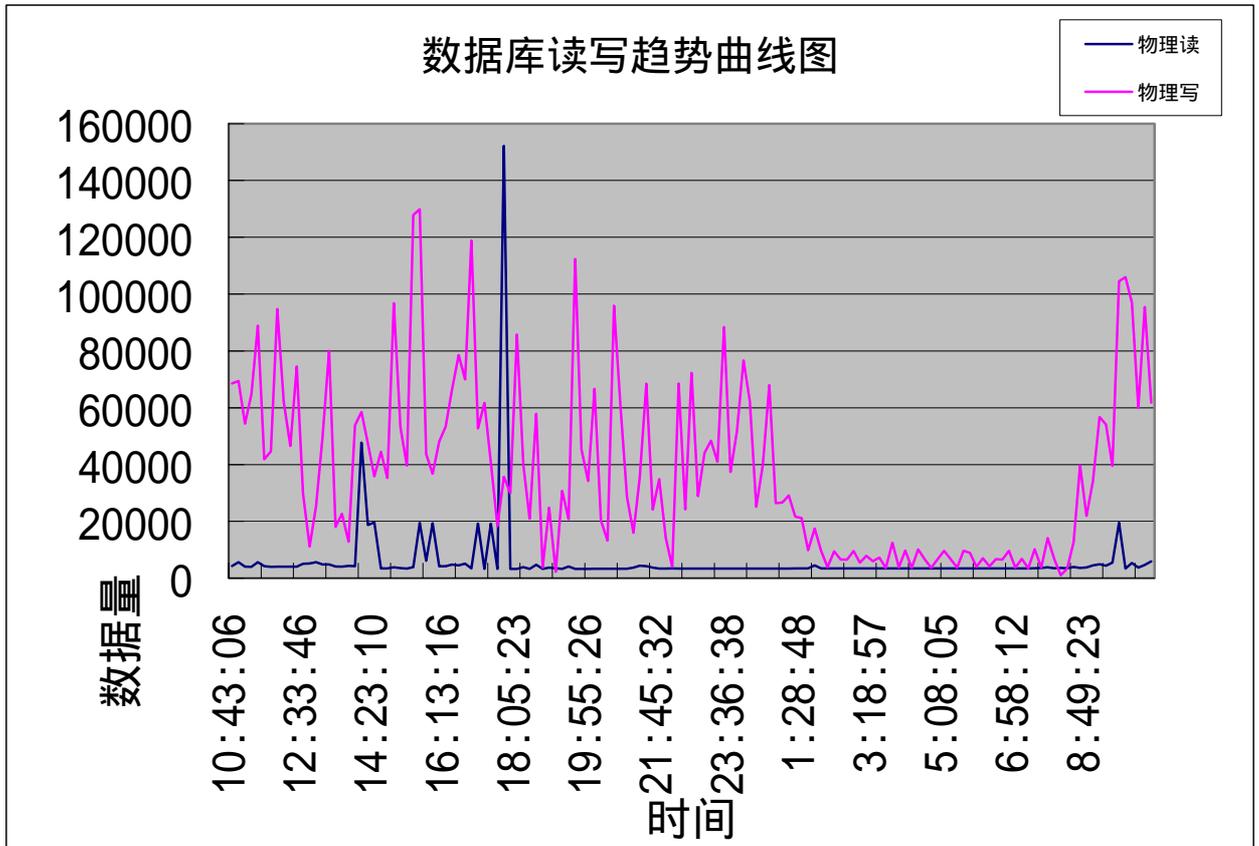
### 1. 物理读写 I/O 操作:

观察物理 I/O 访问,可以看出数据库日常访问的峰值及繁忙程度。

脚本:此脚本按时间生成统计数据(注:以下示例以 8i 为基础,SQL 脚本中引用的 statistic#在不同版本代表的意义可能不同,对于 9i 等版本,你应该修改相应参数值)

```
select
  substr(to_char(snap_time,'yyyy-mm-dd HH24:MI:SS'),12),
  (newreads.value-oldreads.value) reads,
  (newwrites.value-oldwrites.value) writes
from
  perfstat.stats$sysstat oldreads,
  perfstat.stats$sysstat newreads,
  perfstat.stats$sysstat oldwrites,
  perfstat.stats$sysstat newwrites,
  perfstat.stats$snapshot sn
where
  newreads.snap_id = sn.snap_id
and
  newwrites.snap_id = sn.snap_id
and
  oldreads.snap_id = sn.snap_id-1
and
  oldwrites.snap_id = sn.snap_id-1
and
  oldreads.statistic# = 40
and
  newreads.statistic# = 40
and
  oldwrites.statistic# = 41
and
  newwrites.statistic# = 41
and
  (newreads.value-oldreads.value) > 0
and
  (newwrites.value-oldwrites.value) > 0
/
```

图表：



分析：

从趋势图中我们可以看出，数据库每日读操作较为平稳，数据量大约在 4000 左右。在下午 2 点到 5 点期间比较繁忙。峰值达到 150000 左右。

数据库写操作变化也比较平稳，数据改变量在 80000 左右，凌晨一点半到早晨 8 点半左右数据库访问极少。这是一个以写为主的数据库，我们需要更多注意的是写竞争。

## 2. Buffer 命中率

脚本

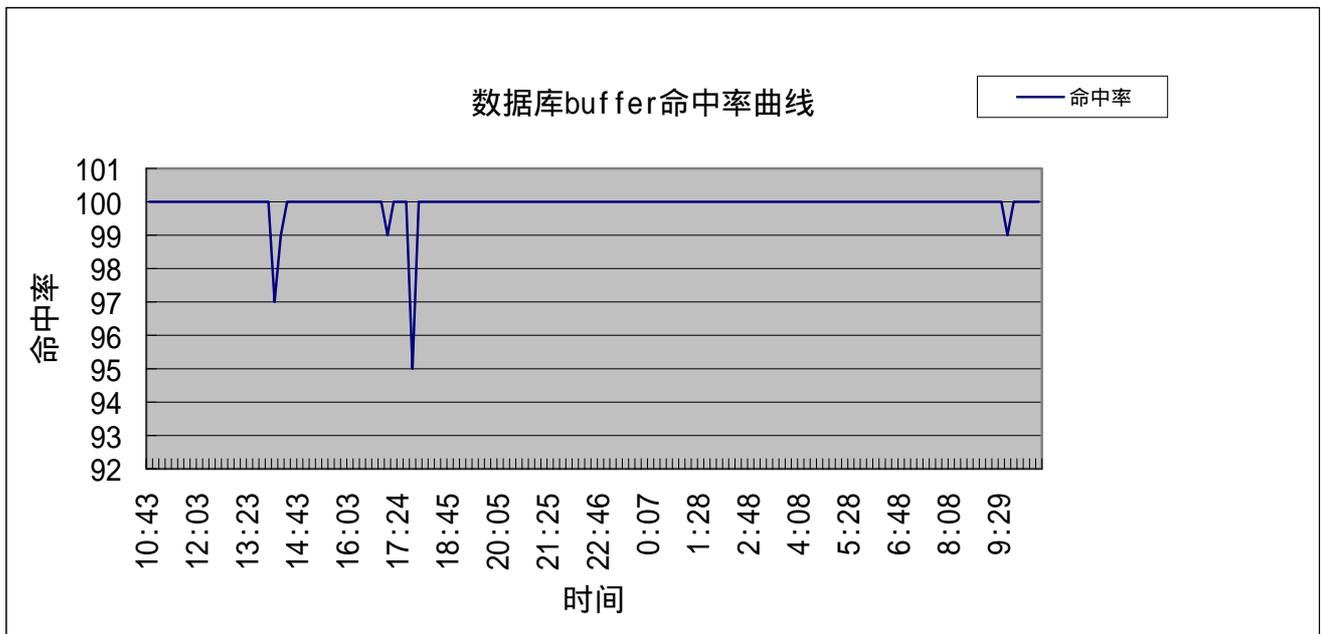
```
select
  substr(to_char(snap_time, 'yyyy-mm-dd HH24:MI'), 12),
  round(100 * (((a.value-e.value)+(b.value-f.value))-(c.value-g.value)) /
  ((a.value-e.value)+(b.value-f.value)))
  "BUFFER HIT RATIO"
from
  perfstat.stats$sysstat a,
  perfstat.stats$sysstat b,
  perfstat.stats$sysstat c,
  perfstat.stats$sysstat d,
  perfstat.stats$sysstat e,
  perfstat.stats$sysstat f,
  perfstat.stats$sysstat g,
  perfstat.stats$snapshot sn
where
  a.snap_id = sn.snap_id
```

```

and
  b.snap_id = sn.snap_id
and
  c.snap_id = sn.snap_id
and
  d.snap_id = sn.snap_id
and
  e.snap_id = sn.snap_id-1
and
  f.snap_id = sn.snap_id-1
and
  g.snap_id = sn.snap_id-1
and
  a.statistic# = 39
and
  e.statistic# = 39
and
  b.statistic# = 38
and
  f.statistic# = 38
and
  c.statistic# = 40
and
  g.statistic# = 40
and
  d.statistic# = 41

```

图表：



分析：

Buffer (buffer hit ratio) 命中率是考察 Oracle 数据库性能的重要指标，它代表在内存中找到需要数据的比率，一般来说，如果该值小于 90%，则可能说明数据库存在大量代价昂贵的 IO 操作，数据库需要调整。

我们数据库的 buffer 命中率几乎接近 100%，最低值在 95% 左右，这个比率是比较优化的。

## 一一.常见等待事件说明

### 什么是等待事件

Oracle的等待事件是衡量Oracle运行状况的重要依据及指标。

等待事件的概念是在Oracle7.0.1.2中引入的,大致有100个等待事件。在Oracle 8.0中这个数目增加到了大约150个,在Oracle8i中大约有200个事件,在Oracle9i中大约有360个等待事件。

主要有两种类别的等待事件,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。

常见的空闲事件有:

- dispatcher timer
- lock element cleanup
- Null event
- parallel query dequeue wait
- parallel query idle wait - Slaves
- pipe get
- PL/SQL lock timer
- pmon timer- pmon
- rdbms ipc message
- slave wait
- smon timer
- SQL\*Net break/reset to client
- SQL\*Net message from client
- SQL\*Net message to client
- SQL\*Net more data to client
- virtual circuit status
- client message

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。

一些常见的非空闲等待事件有:

- db file scattered read
- db file sequential read
- buffer busy waits
- free buffer waits
- enqueue
- latch free
- log file parallel write
- log file sync

## 1. 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描时，基于性能的考虑，数据会分散(scattered)读入 Buffer Cache。如果这个等待事件比较显著，可能说明对于某些全表扫描的表，没有创建索引或者没有创建合适的索引，我们可能需要检查这些数据表已确定是否进行了正确的设置。

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

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

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

我们通过通过一个案例分析来熟悉一下这个等待事件:

| DB Name | DB Id      | Instance | Inst Num | Release   | OPS | Host |
|---------|------------|----------|----------|-----------|-----|------|
| K2      | 1999167370 | k2       | 1        | 8.1.5.0.0 | NO  | k2   |

**这是一个 8.1.5 的数据库系统，通过脚本增强，我们可以在 8.1.5 的数据库上使用 statspack 来进行数据库诊断。**

| Start Id | End Id | Start Time         | End Time           | Snap Length<br>(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

| Event                         | Waits             | Time (cs)        | Wait % Total<br>Wt Time |
|-------------------------------|-------------------|------------------|-------------------------|
| ~~~~~                         |                   |                  |                         |
| <b>db file scattered read</b> | <b>16,842,920</b> | <b>3,490,719</b> | <b>43.32</b>            |
| 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。了解了这些以后我们就可以进一步察看相关 SQL 看是否存在可以优化的 SQL 语句。

SQL ordered by Gets for DB: K2 Instance: k2 Snaps: 170 - 176

| Buffer Gets | Executes | Gets per Exec | % of 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        | 802,918.0     | .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 我们可以获得完整的 SQL 语句。

在这次诊断中，我紧接着去查询的是 v\$session\_longops 数据表，一个分组查询的结果如下：

```

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

```

我们发现这些问题 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！

## 2. db file sequential read-DB 文件顺序读取。

这一事件通常显示与单个数据块相关的读取操作(如索引读取)。

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，可能没有正确的使用驱动表；或者可能说明不加选择地进行索引。

在大多数情况下我们说，通过索引可以更为快速的获取记录，所以对于一个编码规范、调整良好的数据库，这个等待很大是很正常的。

但是在很多情况下，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

能明显快于索引扫描,所以在开发中我们就应该注意,对于这样的查询应该进行避免使用索引扫描。

### 3. Free Buffer-释放缓冲区

这个等待事件表明系统正在等待内存中的可用空间,这说明当前 Buffer 中已经没有 Free 的内存空间。

如果应用设计良好,SQL 书写规范,充分绑定变量,那种等待可能说明 Buffer Cache 设置的偏小,你可能需要增大 DB\_BUFFER\_CACHE。

Free Buffer 等待可能说明 DBWR 的写出速度不够,或者磁盘存在严重的竞争,可以考虑增加检查点、使用更多的 DBWR 进程,或者增加物理磁盘的数量,分散负载,平衡 IO。

### 4. Buffer Busy-缓冲区忙

该等待事件表示正在等待一个以unshareable方式使用的缓冲区,或者表示当前正在被读入buffer cache。一般来说Buffer Busy Wait不应大于1%。

检查缓冲等待统计部分(或V\$WAITSTAT),看一下等待是否位于段头(Segment Header)。如果是,可以考虑增加自由列表(freelist,对于Oracle8i DMT)或者增加freelist groups(在很多时候这个调整是立竿见影的,在8.1.6之前,这个freelists参数不能动态修改;在8.1.6及以后版本,动态修改freelists需要设置COMPATIBLE至少为8.1.6)。

其修改语法为:

```
SQL> alter table sp_item storage (freelists 2);
```

表已更改。

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

如果等待位于undo block上,我们可能需要检查相关应用,适当减少大规模的一致性读取,或者降低一致性读取(consistent read)的表中的数据密度或者增大DB\_CACHE\_SIZE。

如果等待处于data block,可以考虑将频繁并发访问的表或数据移到另一数据块或者进行更大范围的分布(可以增加pctfree值,扩大数据分布,减少竞争),以避开这个"热点"数据块,或者可以考虑增加表中的自由列表或使用本地化管理的表空间(Locally Managed Tablespaces)。

如果等待处于索引块,应该考虑重建索引、分割索引或使用反向键索引。

为了防止与数据块相关的缓冲忙等待,也可以使用较小的块:在这种情况下,单个块中的记录就较少,所以这个块就不是那么"繁忙";或者可以设置更大的pctfree,使数据扩大物理分布,减少记录间的热点竞争。

在执行DML (insert/update/ delete)时 ,Oracle向数据块中写入信息 ,对于多事务并发访问的数据表 ,关于ITL的竞争和等待可能出现 ,为了减少这个等待 ,可以增加initrans ,使用多个ITL槽。

以下是一个生产系统v\$waitstat试图所显示的等待信息:

```
SQL> select * from v$waitstat where count<>0 or time <>0;
```

| CLASS | COUNT | TIME |
|-------|-------|------|
|-------|-------|------|

|            |     |      |
|------------|-----|------|
| data block | 453 | 6686 |
|------------|-----|------|

|             |     |      |
|-------------|-----|------|
| undo header | 391 | 1126 |
|-------------|-----|------|

|            |     |   |
|------------|-----|---|
| undo block | 172 | 3 |
|------------|-----|---|

在 Oracle9i 中 ,引入了一个新概念:ASSM ( Segment Space Management Auto )。通过这个新特性 Oracle 使用位图来管理空间使用。

ASSM 结合 LMT 彻底改变了 Oracle 的存储机制 ,位图 freelist 能够减轻缓冲区忙等待( buffer busy wait ) ,这个问题在 Oracle9i 以前的版本里曾是一个严重的问题。

Oracle 宣称 ASSM 显著地提高了 DML 并发操作的性能 ,因为 ( 同一个 ) 位图的不同部分可以被同时使用 ,这样就消除了寻找剩余空间的串行化。根据 Oracle 的测试结果 ,使用位图 freelist 会消除所有分段头部 ( 对资源 ) 的争夺 ,还能获得超快的并发插入操作

在 Oracle9i 之中 , Buffer Busy wait 不再常见 !

## 5. latch free-latch 释放

latch是一种低级排队机制 ,用于保护SGA中共享内存结构。

latch就像是一种快速地被获取和释放的内存锁。用于防止共享内存结构被多个用户同时访问。如果latch不可用 ,就会记录latch释放失败(latch free miss )。

有两种与latch有关的类型 :

立刻。

可以等待。

假如一个进程试图在立刻模式下获得latch ,而该latch已经被另外一个进程所持有 ,如果该latch不能立即可用的话 ,那么该进程就不会为获得该latch而等待。它将执行另一个操作。

大多数latch问题都与以下操作相关 :

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

通常我们说 ,如果想设计一个失败的系统 ,不考虑绑定变量 ,这一个条件就够了 ,对于异构性

强的系统，不使用绑定变量的后果是极其严重的。

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

当latch miss ratios大于0.5%时，就应当研究这一问题。

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

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

以下我们简单来看一下对栓的查询及跟踪:

```
SQL> select addr,name
 2  from v$latch_children
 3  where name like '%chain%'
 4  /
ADDR          NAME
-----
00000003C0558AE8 enqueue hash chains
00000003C0558A58 enqueue hash chains
00000003C05589C8 enqueue hash chains
00000003C0558938 enqueue hash chains
00000003C05588A8 enqueue hash chains
00000003C0558818 enqueue hash chains
00000003C0558788 enqueue hash chains
00000003C05586F8 enqueue hash chains
00000003C1FC7830 cache buffers lru chain
00000003C1FC7448 cache buffers lru chain
00000003C1FC7060 cache buffers lru chain

ADDR          NAME
-----
00000003C1FC6C78 cache buffers lru chain
00000003C247D970 cache buffers chains
00000003C247C8C0 cache buffers chains
00000003C247B810 cache buffers chains
00000003C247A760 cache buffers chains
00000003C24796B0 cache buffers chains
00000003C2478600 cache buffers chains
00000003C2477550 cache buffers chains
00000003C24764A0 cache buffers chains
00000003C24753F0 cache buffers chains
```

00000003C2474340 cache buffers chains

.....

SQL> col segment\_name for a40

SQL> set linesize 120

SQL> /

```
1  select /*+ ordered */
2     e.owner || '.' || e.segment_name  segment_name,
3     e.extent_id  extent#,
4     x.dbablk - e.block_id + 1  block#,
5     x.tch,
6     l.child#
7  from
8     sys.v$latch_children  l,
9     sys.x$bh  x,
10    sys.dba_extents  e
11  where
12     l.name      = 'cache buffers chains' and
13     l.sleeps   > &sleep_count and
14     x.hladdr   = l.addr and
15     e.file_id = x.file# and
16*    x.dbablk between e.block_id and e.block_id + e.blocks - 1
```

SQL> /

Enter value for sleep\_count: 10000

old 13: l.sleeps > &sleep\_count and

new 13: l.sleeps > 10000 and

| SEGMENT_NAME               | EXTENT# | BLOCK# | TCH   | CHILD# |
|----------------------------|---------|--------|-------|--------|
| -----                      |         |        |       |        |
| SYS.I_ACCESS1              | 14      | 16     | 12    | 1001   |
| SYS.I_DEPENDENCY2          | 2       | 11     | 25    | 804    |
| SYS.I_DEPENDENCY2          | 2       | 13     | 23    | 806    |
| SYS.I_DEPENDENCY1          | 2       | 10     | 1     | 1019   |
| SYS.I_DEPENDENCY1          | 2       | 11     | 2     | 1020   |
| .....                      |         |        |       |        |
| HSCONTENT.HSIDX_INFO_PARAM | 50      | 2      | 35635 | 1017   |
| HSCONTENT.HSIDX_INFO_PARAM | 50      | 3      | 51269 | 1018   |
| HSCONTENT.HSIDX_INFO_PARAM | 50      | 4      | 32415 | 1019   |
| HSCONTENT.HSIDX_INFO_PARAM | 50      | 5      | 51956 | 1020   |
| HSCONTENT.HSIDX_INFO_PARAM | 50      | 6      | 57509 | 1021   |
| HSCONTENT.PK_HS_DLF_OBJECT | 11      | 2      | 4434  | 809    |
| HSCONTENT.PK_HS_DLF_OBJECT | 11      | 6      | 5738  | 813    |
| HSCONTENT.PK_HS_DLF_OBJECT | 14      | 2      | 524   | 1001   |
| HSCONTENT.PK_HS_DLF_OBJECT | 23      | 2      | 4843  | 1001   |
| HSCONTENT.PK_HS_DLF_OBJECT | 24      | 2      | 4361  | 1001   |
| HSCONTENT.PK_HS_DLF_OBJECT | 25      | 13     | 2507  | 804    |

.....

## 6. Enqueue.

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，如记录中的数据，以避免两个人在同一时间更新同一数据。enqueue包括一个排队机制，即FIFO（先进先出）排队机制。

Enqueue等待常见的有ST、HW、TX、TM等

ST enqueue，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。

我们通过一个实例来看一下：

```
DB Name      DB Id  Instance  Inst Num Release  OPS Host
-----
DB           40757346  aaa       1 8.1.7.4.0 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。

```
Cache Sizes
~~~~~
db_block_buffers: 51200      log_buffer: 2097152
db_block_size:   16384    shared_pool_size: 209715200
.....
Top 5 Wait Events
~~~~~
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。我们来关注一下enqueue等待事件，在89.82 (mins)的采样间隔内，累计enqueue等待长达16,192,686cs,即45小时左右。这个等待已经

太过显著，实际上这个系统也因此遭遇了巨大的困难，观察到队列等待以后，我们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，我们看到以下详细内容：

Enqueue activity for DB: DB Instance: aaa Snaps: 2716 -2718

-> ordered by waits desc, gets desc

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

我们看到主要队列等待在等待ST锁定，对于DMT，我们说这个等待跟FET\$, UET\$的争用紧密相关。我们在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     | 78.6          | 8.5     | 528349613  |
| delete from uet\$ where ts#=:1 and segfile#=:2 and segblock#=:3 and 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 where t.ts#=f.ts# and t.dflxtpct!=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  |            |               |         |            |

我们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，我们已经可以准确的为该系统定位问题，相应的解决方案也很容易确定，在8.1.7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。

HW enqueue指和段的高水位标记相关等待；手动分配适当区可以避免这一等待。

TX是最常见的enqueue等待。TX enqueue等待通常是以下三个问题之一产生的结果。

第一个问题是唯一索引中的重复索引，你需要执行提交（commit）/回滚（rollback）操作来释放enqueue。

第二个问题是对同一位图索引段的多次更新。因为单个位图段可能包含多个行地址（rowid），所以当多个用户试图更新同一段时，可能一个用户会锁定其他用户请求的记录，这时等待出现。直到获得锁定的用户提交或回滚，enqueue释放。

第三个问题，也是最可能发生的问题是多个用户同时更新同一个块。如果没有足够的ITL槽，就会发生块级锁定。通过增大initrans和/或maxtrans以允许使用多个ITL槽（对于频繁并发进行DML操作的数据表，在建表之初就应该考虑为相应参数设置合理的数值，避免系统运行以后在线的更改，在8i之前，freelists等参数不能在线更改，设计时的考虑就尤为重要），或者增大表上的pctfree值，就可以很容易的避免这种情况。

TM enqueue队列锁在进行DML操作前获得，以阻止对正在操作的数据表进行任何DDL操作（在DML操作一个数据表时，其结构不能被更改）。

## 7. Log Buffer Space-日志缓冲空间

当你将日志缓冲（log buffer）产生重做日志的速度比LGWR的写出速度快，或者是当日志切换（log switch）太慢时，就会发生这种等待。

这个等待出现时，通常表明redo log buffer过小，为解决这个问题，可以考虑增大日志文件的大小，或者增加日志缓冲器的大小。

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

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       |           |                |                 |
|-------------------------|-----------|----------------|-----------------|
| Event                   | Waits     | Wait Time (cs) | % Total Wt Time |
| log file parallel write | 1,436,993 | 1,102,188      | 10.80           |

|                                    |                  |                |             |
|------------------------------------|------------------|----------------|-------------|
| <b>log buffer space</b>            | <b>16,698</b>    | <b>873,203</b> | <b>8.56</b> |
| <b>log file sync</b>               | <b>1,413,374</b> | <b>654,587</b> | <b>6.42</b> |
| <b>control file parallel write</b> | <b>329,777</b>   | <b>510,078</b> | <b>5.00</b> |
| <b>db file scattered read</b>      | <b>425,578</b>   | <b>132,537</b> | <b>1.30</b> |
| -----                              |                  |                |             |

## 8. Log File Switch-日志文件切换

当这个等待出现时，表示所有的提交(commit)的请求都需要等待"日志文件切换"的完成。

Log file Switch 主要包含两个子事件:

```
log file switch (archiving needed)
log file switch (checkpoint incomplete)
```

```
log file switch (archiving needed)
```

这个等待事件出现时通常是因为日志组循环写满以后，第一个日志归档尚未完成，出现该等待。出现该等待，可能表示 io 存在问题。

解决办法：

- 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 .

```
log file switch (checkpoint incomplete)-日志切换（检查点未完成）
```

当你的日志组都写完以后，LGWR 试图写第一个 log file，如果这时数据库没有完成写出记录在第一个 log file 中的 dirty 块时（例如第一个检查点未完成），该等待事件出现。

该等待事件通常表示你的 DBWR 写出速度太慢或者 IO 存在问题。

为解决该问题，你可能需要考虑增加额外的 DBWR 或者增加你的日志组或日志文件大小。

## 9. log file sync-日志文件同步

当一个用户提交或回滚数据时，LGWR 将会话期的重做由日志缓冲器写入到重做日志中。日志文件同步过程必须等待这一过程成功完成。

为了减少这种等待事件，可以尝试一次提交更多的记录（频繁的提交会带来更多的系统开销）。

将重做日志置于较快的磁盘上，或者交替使用不同物理磁盘上的重做日志，以降低归档对 LGWR 的影响。

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

## 10. log file single write

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

#### 11. log file parallel write

从 log buffer 写 redo 记录到 redo log 文件，主要指常规写操作(相对于 log file sync)。

如果你的 Log group 存在多个组成员，当 flush log buffer 时，写操作是并行的，这时候此等待事件可能出现。

尽管这个写操作并行处理，直到所有 I/O 操作完成该写操作才会完成(如果你的磁盘支持异步 IO 或者使用 IO SLAVE，那么即使只有一个 redo log file member,也有可能出现此等待)。

这个参数和 log file sync 时间相比较可以用来衡量 log file 的写入成本。通常称为同步成本率。

#### 12. control file parallel write-控制文件并行写

当 server 进程更新所有控制文件时，这个事件可能出现。

如果等待很短，可以不用考虑。如果等待时间较长，检查存放控制文件的物理磁盘 I/O 是否存在瓶颈。

多个控制文件是完全相同的拷贝，用于镜像以提高安全性。对于业务系统，多个控制文件应该存放在不同的磁盘上，一般来说三个是足够的，如果只有两个物理硬盘，那么两个控制文件也是可以接受的。在同一个磁盘上保存多个控制文件是不具备实际意义的。

减少这个等待，可以考虑如下方法：

- 减少控制文件的个数(在确保安全的前提下)
- 如果系统支持，使用异步 IO
- 转移控制文件到 IO 负担轻的物理磁盘

#### 13. control file sequential read/ control file single write 控制文件连续读/控制文件单个写

对单个控制文件 I/O 存在问题时，这两个事件会出现。

如果等待比较明显，检查单个控制文件，看存放位置是否存在 I/O 瓶颈。

使用查询获得控制文件访问状态：

```
select P1 from V$SESSION_WAIT
where EVENT like 'control file%' and STATE='WAITING';
```

解决办法：

移动有问题的控制文件到快速磁盘

如果系统支持，启用异步 I/O

#### 14. direct path write-直接路径写

该等待发生在，系统等待确认所有未完成的异步 I/O 都已写入磁盘。

对于这一写入等待，我们应该找到 I/O 操作最为频繁的数据文件(如果有过多的排序操作，很有可能就是临时文件)，分散负载，加快其写入操作。

如果系统存在过多的磁盘排序，会导致临时表空间操作频繁，对于这种情况，可以考虑使用 Local 管理表空间，分成多个小文件，写入不同磁盘或者裸设备。

我们可以看一个 report 的典型例子：

| DB Name  | DB Id     | Instance            | Inst Num | Release             | OPS | Host         |
|----------|-----------|---------------------|----------|---------------------|-----|--------------|
| DB       | 294605295 | db                  | 1        | 8.1.5.0.0           | 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 |     | <b>21.30</b> |

这是一个 20 分钟的采样报告

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

我们注意到在 Top 5 等待事件中，最为显著的等待事件就是 direct path write。  
基于此，我们继续向下追查相关排序部分统计数据：

| Statistic   | Total | per Second | per Trans |
|---|-------|------------|-----------|
| Instance Activity Stats for DB: DB Instance: db Snaps: 65 - |       |            |           |
| .....   |       |            |           |

|                |           |         |          |
|----------------|-----------|---------|----------|
| 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,对于这种情况,我们可以很容易的猜测到,临时表空间的读写操作肯定相当频繁:

File IO Statistics for DB: GHCXSDB Instance: ghcxldb Snaps: 65 - 66

| Tablespace  | Filename       |             |             |                |                         |
|-------------|----------------|-------------|-------------|----------------|-------------------------|
|             | Reads          | Avg Blks Rd | Avg Rd (ms) | Writes         | Tot Waits Avg Wait (ms) |
| PERFSTAT    | 88             | 1.0         | 12.5        | 821            | 0                       |
| RBS         | 7              | 1.0         | 0.0         | 1,399          | 0                       |
| SYSTEM      | 17             | 1.0         | 11.8        | 50             | 0                       |
| <b>TEMP</b> | <b>223,152</b> | <b>1.5</b>  | <b>0.2</b>  | <b>371,303</b> | <b>0</b>                |

对于这种情况,我们建议,可以适当增加 sort\_area\_size 的大小,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。有一个测试数字可以参考:磁盘排序的时间大约是内存排序的 14000 倍。

### 15. slave wait-从属进程等

Slave Wait 是 Slave I/O 进程等待请求,是一个空闲参数,一般不说明问题。

### 16. Idle Event-空闲事件

最后我们来看几个空闲等待事件。

一般来说,空闲等待是指系统因为无事可做的等待,或者等待用户的请求或响应等,通常我们可以忽略这些等待事件。

空闲事件可以通过 stats\$idle\_event 表查询得到。

我们看一下系统的主要空闲等待事件,对这些事件大家应该有个大致的印象,如果你的 Top 5 等待事件中,主要都是这些事件,那么一般来说你的系统是比价清闲的:

```
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.
```

## 一二. 在 815 上的安装配置

### 数据库状况

```
SQL> select * from v$version;

BANNER
-----
Oracle8i Enterprise Edition Release 8.1.5.0.0, 64 bit - Producti
PL/SQL Release 8.1.5.0.0 - Production
CORE Version 8.1.3.0.0 - Production
TNS for HPUX: Version 8.1.5.0.0 - Production
NLSRTL Version 3.4.0.0.0 - Production
```

### 运行 statscbps.sql

```
SQL> @statscbps.sql

View created.
Synonym created.
Grant succeeded.
SQL>
```

### 创建表空间

```
SQL> select name from v$datafile;

NAME
-----
/u03/oradata/ksec2/system01.dbf
.....
/u03/oradata/ksec2/temp03.dbf
/u03/oradata/ksec2/capp_add1.dbf

16 rows selected

SQL> create tablespace perfstat
  2 datafile '/u03/oradata/ksec2/perfstat01.dbf'
  3 size 200M;

Tablespace created

SQL>
```

## 运行 statscre.sql 创建 Stats 对象

```
Specify PERFSTAT user's default tablespace
Enter value for default_tablespace: perfstat
Using perfstat for the default tablespace

User altered.

User altered.

Specify PERFSTAT user's temporary tablespace
Enter value for temporary_tablespace: temp
Using temp for the temporary tablespace
User altered.
....
Creating Package STATSPACK...

Package created.

No errors.
Creating Package Body STATSPACK...

Package body created.

No errors.

NOTE:
STATSPACK complete. Please check statspack.lis for any errors.
```

## 测试安装

```
SQL> execute statspack.snap;
PL/SQL procedure successfully completed.
```

## 运行 job 收集数据

```
SQL> show user
USER is "PERFSTAT"
SQL> @statsauto.sql

PL/SQL procedure successfully completed.

Job number for automated statistics collection for this instance
~~~~~
```

Note that this job number is needed when modifying or removing the job:

```
JOBNO
```

```
-----
```

```
1
```

Job queue process

```
-----
```

Below is the current setting of the job\_queue\_processes init.ora parameter - the value for this parameter must be greater than 0 to use automatic statistics gathering:

```
NAME                                TYPE    VALUE
```

```
-----
```

```
job_queue_processes                integer 10
```

Next scheduled run

```
-----
```

The next scheduled run for this job is:

```
JOB NEXT_DATE NEXT_SEC
```

```
-----
```

```
1 20-FEB-03 16:00:00
```

查看 job

```
SQL> select JOB , NEXT_DATE from user_jobs;
```

```
JOB NEXT_DATE NEXT_SEC
```

```
-----
```

```
1 2003-2-20 1 16:00:00
```