DIAGNOSING PERFORMANCE USING STATSPACK

Connie Dialeris and Graham Wood, Server Technologies, Oracle Corporation

INTRODUCTION TO STATSPACK

Statspack is a performance diagnosis tool, first shipped with Oracle8i Release 8.1.6 and enhanced in 8.1.7. Although shipped with 8.1.6 and beyond, scripts are available that will allow the 8.1.7 version to run on any system running Oracle8 or above. In this paper we will concentrate on the 8.1.7 functionality.

Statspack is primarily a diagnostic tool for instance-wide performance problems; it also supports application tuning activities by identifying high-load SQL statements. It can be used both proactively to monitor the changing load on a system, and also reactively to investigate a performance problem. To use Statspack you take a number of 'snapshots' of the Oracle performance data and you can then report on any pair of these snapshots. The greatest benefits are seen when there is 'baseline' performance data available for the system to compare with current data. This will allow you to easily answer questions like 'Has the throughput of my system increased?' and 'Have the resources used per transaction on my system increased?' or 'What has *really* changed?'

COMPARISON WITH BSTAT/ESTAT

The BSTAT/ESTAT scripts have been shipped with the Oracle RDBMS since Oracle V6. In this time they have become a commonly used tool for helping to diagnose database performance problems. The scripts were simple to use, and, because they are shipped with the server, always available at every site. They did however have some limitations. Each run of BSTAT/ESTAT produced a report and then dropped all the tables that had been used for collecting the data. This meant that it was not possible go back at a later stage and perform further analysis. The capture and reporting were totally geared towards database instance tuning whereas the biggest improvements on a system are normally to be made by application or schema optimizations.

Statspack has been designed to keep the simplicity of the old scripts while offering much more functionality.

One difference between BSTAT/ESTAT and Statspack is how the data is collected. When using BSTAT/ESTAT there is no way to separate the collection of data from the report creation; when ESTAT is run a report is automatically produced and then the data tables are dropped, losing the data just captured. With STATSPACK, these two phases are totally distinct. Data collection is performed either on demand, or can be automated, and is stored in permanent tables. The viewing of the data collection is at the hands of the performance practitioner.

Separating these two phases and storing the data in permanent tables allows the report to be based on any pair of data points - for example it may be reasonable for the DBA to use the supplied automation script to automate data collection every hour on the hour. If at some later point a performance issue arose which may be better investigated by looking at a three hour data window, rather than an hour's worth of data, the only thing the DBA need do is specify the required start point and end point when running the report.

The report produced by Statspack look very different to a ESTAT report.txt. The first page is designed to display all the most important data about the system, and will help direct where to look next to analyze performance. A sample summary page is shown in Listing 2.

As well as collection and reporting, Statspack also contains files that allow the purging of data that is no longer required, and a sample export file to allow data to be moved to another system for archival or analysis.

A summary of the differences between Statspack and BSTAT/ESTAT is given below in Table 1.

Feature	Available in Statspack	Available in BSTAT/ESTAT
Instance Summary Page	Y	-
Normalization of instance statistics by time and number of transactions	Y	-
Wait Events	Y	Y
High Resource SQL	Y	-
Instance Activity Statistics	Y	Y
Tablespace and File IO Statistics	Y	Y
Buffer Wait Breakdown by Type	Y	Y
Enqueue statistics	Y	-
Rollback Segment Activity and Storage data	Y	Y
Latch Activity	Y	Y
Latch Sleep breakdown	Y	-
Latch Children + Latch Parent	Y*	-
Buffer Pool Statistics	Y	-
Dictionary Cache Activity	Y	Y
Library Cache Activity	Y	Y
SGA Memory Summary	Y	-
SGA Memory Breakdown	Y	-
Non-default init.ora parameters	Y	Y
Configurable output name	Y	-
Able to move performance data	Y	-
Configurable amount of data collected	Y	-
Able to run in multiple instances for OPS	Y	-

* When Statspack level 10 data collection is used

Table 1. Comparison of Statspack and BSTAT/ESTAT.

USING STATSPACK

INSTALLING STATSPACK

Run the installation script using SQL*Plus from within the $ORACLE_HOME/rdbms/admin directory or the equivalent on your system:

```
SQL> connect / as sysdba
SQL> @spcreate
```

This will create a database user PERFSTAT and the Statpack schema. Note that the script *must* be run from SQL*Plus. All subsequent Statspack operations are run as the PERFSTAT user.

COLLECTING DATA WITH STATSPACK

Once you have installed Statspack the simplest way to collect a 'snapshot' of performance data is by executing the snap function from the Statspack package.

```
SQL> execute statspack.snap;
```

The volume of data collected can be varied by setting a number of parameters; there is a trade off between the time taken and intrusion effects of performing the snapshots, and the level to which subsequent analysis can be performed.

The default level of collection, level 5, is adequate for most applications. At this level the normal performance statistics are captured along with the high-resource-usage SQL statements. Parameters can be used to set the limits for the SQL statement collection, which will be highly system dependent.

It is also possible to capture statistics from an individual session as part of a snapshot by using the i_session_id parameter to the procedure. The example below will capture session level statistics for the session with a session id (Oracle sid) of 32.

```
SQL> execute statspack.snap(i_session_id=>32);
```

Data collection can also be automated at either the OS or database level using the `dbms_job` package. An example of the latter is provided in the `spauto.sql` file.

HOW OFTEN TO COLLECT

No matter how your system is performing, if you don't have existing baseline data the time to begin gathering baseline snapshots is *now*. You can automate taking a snapshot at repeated intervals such as once an hour every day, or you can take a few representative snapshots at different peak times on a regular basis. For example, your baseline may include a snapshot every hour between 10am and 1pm, then another snapshot at 6pm, and include two additional snapshots to monitor the batch window of midnight to 6am. If you capture a large number of snapshots it is possible to purge unneeded data using the `sppurge.sql` script.

It is recommended to set the parameter `timed_statistics` to true for your instance, as setting this parameter provides timing data which is invaluable for performance tuning, more than outweighing any additional overhead.

PRODUCING REPORTS WITH STATSPACK

Once multiple snapshots have been taken, it is possible to run reports to examine the activity on the instance by running the supplied report. To examine the delta in statistics between two times, run the `spreport.sql` report while being connected to the PERFSTAT user.

You will be prompted for:

1. The beginning snapshot Id
2. The ending snapshot Id
3. The name of the report output file to be created (default name includes the begin and end snapshot id)

A sample report generation is given below in Listing 1.

```
SQL>  connect perfstat/perfstat
Connected.
SQL>  @spreport

        DB Id DB Name     Instance# Instance
  ----------- ---------- ---------- ----------
   1361567071 MAIL               1   MAIL

  Completed Snapshots

  Instance   DB Name    SnapId      Snap Started      Snap Level
  ---------- ---------- ------ --------------------- ----------
  MAIL       MAIL            1  17 Aug 2000 10:00:16          5
                             2  17 Aug 2000 12:00:28          5

  Enter beginning Snap Id: 1
  Enter ending    Snap Id: 2

  Enter name of output file [sp_11_12] : <enter name or return>
```

ANALYZING STATSPACK REPORTS

The bulk of the data in the report is presented in a manner that will be familiar to those who have used BSTAT/ESTAT. In this paper we will concentrate on the two main area that are different in the Statspack report, the Summary page and SQL reporting.

The Statspack report is designed to be used in a top down manner. The hope is that you will not need to read 90% of the report that is produced. The Summary page of the report gives an overview of the database operations and should be used as the basis of 'directed drilldown' rather than requiring the whole report to be reviewed. An example first page from a Statspack report taken from a production database appears below in Listing 2.

```
STATSPACK report for

DB Name         DB Id    Instance     Inst Num Release     OPS Host
------------ ----------- ------------ -------- ----------- --- ------------
MAIL          1361567071 MAIL                1 8.1.6.0.0   YES gm01

                Snap Id     Snap Time      Sessions
                ------- ------------------ --------
 Begin Snap:          1 17-Aug-00 10:00:16      830
   End Snap:          2 17-Aug-00 12:00:28      830
```

```
    Elapsed:                120.20 (mins)

Cache Sizes
~~~~~~~~~~~
           db_block_buffers:    1113600          log_buffer:    4194304
              db_block_size:       8192    shared_pool_size:  419430400

Load Profile
~~~~~~~~~~~~                            Per Second       Per Transaction
                                   ---------------       ---------------
                  Redo size:          1,988,545.66             13,410.75
              Logical reads:             38,259.37                258.02
              Block changes:              7,624.61                 51.42
             Physical reads:                497.22                  3.35
            Physical writes:              1,570.41                 10.59
                 User calls:              1,840.56                 12.41
                     Parses:                224.24                  1.51
                Hard parses:                  0.10                  0.00
                      Sorts:                188.84                  1.27
                     Logons:                  0.18                  0.00
                   Executes:              2,086.71                 14.07
               Transactions:                148.28

  % Blocks changed per Read:   19.93    Recursive Call %:   47.62
 Rollback per transaction %:    1.50       Rows per Sort:    5.20

Instance Efficiency Percentages (Target 100%)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
            Buffer Nowait %:   98.39       Redo NoWait %:  100.00
            Buffer  Hit   %:   98.70    In-memory Sort %:   99.82
            Library Hit   %:  100.03        Soft Parse %:   99.96
         Execute to Parse %:   89.25         Latch Hit %:   99.46
Parse CPU to Parse Elapsd %:   72.14     % Non-Parse CPU:  100.00

 Shared Pool Statistics        Begin   End
                               ------  ------
             Memory Usage %:   81.71   83.10
    % SQL with executions>1:   34.74   33.47
  % Memory for SQL w/exec>1:   66.93   66.29

Top 5 Wait Events
~~~~~~~~~~~~~~~~~                                                 Wait     % Total
Event                                            Waits  Time (cs)   Wt Time
-------------------------------------------- ------------ ------------ -------
buffer busy waits                               4,452,795   11,794,858   38.94
db file sequential read                         3,318,246    6,355,852   20.98
log file sync                                   1,050,639    5,205,858   17.19
latch free                                      2,708,524    3,559,982   11.75
enqueue                                            88,962    1,227,221    4.05
```

Listing 2. A sample Statspack Summary page.

THE SUMMARY PAGE

ENVIRONMENT SECTION

At the top of each Statspack report summary page are details of the environment and the two snapshots used for the report. The major memory areas of the system, e.g. database buffer and shared pool are also itemized. This section gives a quick overview of the system that is being examined and allows you to see major changes like Oracle software version changes or SGA size changes from a baseline report.

LOAD PROFILE

The load profile section will allow you to determine if the load is changing over time when compared to a baseline. If an application is stable then the per transaction ratios presented here should not change dramatically over time. Similarly, if the per second ratios are the same then the rate at which the system is processing load has not changed. When looking at a report in isolation the load profile can be used to characterize the application load. For most of the ratios there are no *correct* values, however logon rates of more than one or two per second, hard parse rates of more than 100 per second and overall parse rates of over 300 per second would warrant closer attention.

INSTANCE EFFICIENCY

This section contains most of the well know ratios related to the operation of the database instance such as the Buffer Hit Ratio (also called the Cache Hit Ratio) and the Library Hit ratio (also known as the Library Cache Hit ratio). As in the Load Profile section there are no *correct* values, only those that are appropriate for your application and workload. In a DSS environment, performing large parallel queries utilizing direct reads a Buffer Hit Ratio of 20% may be perfectly acceptable, while the same ratio in an OLTP system would be totally unacceptable. Comparing the Instance Efficiency ratios over time allows you to see trends, and investigate them, before they become major problems.

SHARED POOL STATISTICS

The data presented in the Shared Pool section will quickly show how much of the shared pool is currently free (on a system that has been running for some time this would perhaps be better presents as shared pool *wasted*.) and shows how much of the shared pool is in use by statements that have only been executed a single time. Large numbers of single use statements probably indicate the use of literal values in the application and the resulting additional parsing requirements can become the bottleneck of the system.

TOP 5 WAIT EVENTS

The final section of the Summary page shows the Top 5 Wait Event on the database for the period of the report. Idle events, such as '`SQL*Net message from client`', are not included in this list. The top events are listed in descending order of wait time when `timed_statistics` is true for the database, which allows us to show the percentage wait time attributable to each event. This is normally the section from which the drilldown into more detailed data will be performed. When tuning we are always looking for the biggest payback from our efforts and in the case of our sample report it can be seen that almost 39% of all wait time was spent waiting for the '`buffer busy wait`' event, where as reads accounted for only 21%. In this case we would drill down to the Buffer Waits section of the report and the File and Tablespace IO sections to identify which files were causing the problem.

If the top wait event was an IO event we would drill down to the SQL statement data, ordered by physical reads to see which statements were performing large numbers of IOs, and to the Tablespace and IO sections of the report to look for any files with slow response times.

High latch waits in this section obviously lead us to look at the more detailed latch statistics to determine which particular latches are causing problems.

HIGH LOAD SQL

For performance issues which are isolated to a particular application or program, the best course of action is to examine the application code, the SQL issued and the parameters or actions specified by the user. It is also possible

that a single resource intensive operation on the system can impact all users of the system.
You can use Statspack to identify high-load SQL if a level 5 (or above) snapshot is taken, as the SQL text and resource usage is captured and stored in the `stats$sql_summary` table. The top SQL statements (according to buffer gets, physical reads, executions, memory usage and version count) are then displayed in the Statspack report. Listing 3 below show a sample output ordered by buffer gets. Note that the report shows what percentage of the total load a statement contributed. A single statement accounts for 10% of all buffer gets in the reporting period in our example. On a system that is CPU bound major performance gains can often be made by tuning the statements performing the most buffer gets. If IO is the predominant wait event the target for tuning should be the source of the most physical IOs. Although only the first few lines of the SQL statements are output in the report, the full text of the statement can be retrieved using the reported hash value in the following statement:

```
select sql_text
  from stats$sqltext
 where hash_value = &hash_value
 order by piece;
```

```
Buffer Gets    Executions  Gets per Exec  % Total  Hash Value
-------------- ------------ -------------- ------- ------------
     27,687,341       34,404          804.8    10.0   1708631997
select folder_id, folder_type,  disp_name, f_access, to_char(mod
ify_date + (:1 / 1440), 'DD Mon YYYY HH24:MI:SS')  from om_folde
r_list  where parent_id  + 0  = :2 and        folder_type in (0,
 1, 3, 5, 6, 9, 10, 12)  order by folder_name

     17,655,131    1,125,746           15.7     6.4   3679584379
insert into om_ext_header (msg_id,   order_no, prompt, value, eh
_type, info1, info2, info3) values   (:1, :2, :3, :4, :5, :6, :7
, :8)

     15,219,293          764       19,920.5     5.5   1724912937
SELECT msg_id, rowid FROM om_instance WHERE queue = 'D'        F
OR UPDATE skip locked

     13,652,460      148,038           92.2     4.9   3911477900
BEGIN mail_srvr.LOCAL_DELIVERY2(:1, :2, :3,  :4, :5, :6, :7); EN
D;

     12,621,633        1,888        6,685.2     4.6   2789544723
BEGIN mail_api.delete_fldr(:1, :2); END;

     12,046,692      429,545           28.0     4.4   2436931928
insert into om_datae (msg_id, part, order_no,  data_line, line2,
 line3, line4, line5, line6, line7, line8, line9, line10)  value
s (:1, :2, :3, :4, :5, :6, :7, :8, :9, :10, :11, :12, :13)

      9,180,196        1,143        8,031.7     3.3    552417555
DELETE FROM OM_EXT_HEADER WHERE MSG_ID = :b1
```

Listing 3. Sample Top SQL output

SUMMARY

Statspack succeeds the BSTAT/ESTAT scripts and greatly improves on their functionality. The enhanced reporting and the addition of SQL level data simplify the task of diagnosing performance problems on Oracle systems,

especially when used as part of a performance method . The use of the database to store performance data also serves as a basis for building a repository of performance data from a system which could be used for trending and capacity management in the future.

To find out more about Statspack read the file spdoc.txt, which is shipped with the product, and also go to:

`http://technet.oracle.com/deploy/performance`

For information on running Statspack against earlier releases follow the links from:

`http://www.oracle.com/oramag/oracle/00-Mar/o20tun.html`