OPENTURBO Performance Audit Report
ORACLE
March 14, 2005

Introduction

The IMAXSOFT benchmark environment has a 2-node 9iRAC environment running Oracle Enterprise 9.2.0.1 64-bit RDBMS. The nodes are configured identically with 15GB RAM, 2 CPU's, and a fiber-switch connection to a HP EVA SAN device with a single shared volume consisting of 14 physical drives. The database environment was essentially created following the guidelines set forth by Oracle documentation, and many defaults were taken if proper values were not known for configuration parameters. However, all setup was done using best practices at the minimum.

A third party independent firm was tasked with performing a thorough analysis of the environment to determine if any performance improvements could be made, or find any problems within the RAC environment itself. The independent firm provided a list of findings and recommendations that were performed to IMAXSOFT.

The IMAXSOFT OPENTURBO 2.0 made significant performance improvements to their core, which improved performance 2 to 3 times. However, there is still a desire for better performance, and a lingering feeling that the database is still the culprit of the performance problems.

The independent firm has now been tasked to perform a more in depth, thorough analysis of the database by taking statistical measurements of a batch job, and capture information about its activities in real time.

Methodology

Using the independent firm best practices in Performance Tuning, and respecting time/budgetary constraints, the following tasks were performed to gather and analyze the data to make proper recommendations:

1. Database statistical snapshots – memory utilization, wait statistics, latch contention, I/O utilization, CPU utilization, parameter settings, object statistics. This was done using the Oracle Statspack utility that took snapshots of all database statistics in 15 minute increments during the run of the batch job.
2. Server utilization – top, sar, and vmstat.
3. Session traces – This was generated using Oracle Trace utilities and then parsed using the Oracle TKPROF utility to format and aggregate the statistics of the batch job run.

Findings

Batch Job Session Trace

NOTE: Please reference tkprof_20040730.out for referencing statistics

This report contains information regarding all of the SQL that was performed by a given session. Here is a brief definition of the fields contained within this report:

SQL Trace Facility Statistics:

TKPROF lists the statistics for a SQL statement returned by the SQL trace facility in rows and columns. Each row corresponds to one of three steps of SQL statement processing:

- PARSE

This step translates the SQL statement into an execution plan.
This includes checks for proper security authorization and checks or the existence of tables, columns, and other referenced objects.

- EXECUTE

This step is the actual execution of the statement by Oracle.
For INSERT, UPDATE, and DELETE statements, this step modifies the data. For SELECT statements, the step identifies the selected rows.

- FETCH

This step retrieves rows returned by a query.
Fetches are only performed for SELECT statements.

The step for which each row contains statistics is identified by the value of the call column. The other columns of the SQL trace facility output are combined statistics for all parses, all executes, and all fetches of a statement:

COUNT: Number of times a statement was parsed, executed, or fetched.

CPU: Total CPU time in seconds for all parse, execute, or fetch calls for the statement.

ELAPSED: Total elapsed time in seconds for all parse, execute, or fetch calls for the statement.

DISK: Total number of data blocks physically read from the datafiles on disk for all parse, execute, or fetch calls.

QUERY: Total number of buffers retrieved in consistent mode for all parse, execute, or fetch calls. Buffers are usually retrieved in consistent mode for queries.

CURRENT: Total number of buffers retrieved in current mode. Buffers are often retrieved in current mode for INSERT, UPDATE, and DELETE statements.

+ The sum of QUERY & CURRENT is the total number of buffers accessed.

ROWS: Total number of rows processed by the SQL statement. This total does not include rows processed by sub-queries of the SQL statement.

This report summarizes ALL of the activities of this batch job, so there is nothing in here that is missing, nor unrelated to the batch job.

I would like to go through each of the captured queries and explain what is being reported:

Here is the first captured query and its related statistics:

SELECT
DIST_ID, SPONSOR_ID, APPL_NAME, COUNTRY_CODE, ALLERGIC_FLAG, COUNT_FLAG, APPL_DT, STATUS, RANK, GSV_ACCUM, GSV_ERR_ALLOW_FG, GSV_AMT_02, GSV_AMT_03, GSV_AMT_04, GSV_AMT_05, GSV_AMT_06, GSV_AMT_07, GSV_AMT_08, GSV_AMT_12, GSV_AMT_13, PSV_ACCUM, PSV_ERR_ALLOW_FG, PSV_ERR_ALLOW_MO, PSV_AMT_02, PSV_AMT_03, PSV_AMT_04, PSV_AMT_05, PSV_AMT_06, PSV_AMT_07, PSV_AMT_08, PSV_AMT_12, PSV_AMT_13, REBATE_AMT, MTD_1099_AMT,

NUM_MO_INACTIVE, GRP_DEV_BON_AMT, PRES_PNTS_MTD, NUM_OF_NEW_DIST, PREV_AR_BAL, PREV_AR_BAL_CN, DIST_FLAG2, TRNS_SV, INV_SV, PSV_BONUS, NUM_OF_NEW_ASST, NEW_TRAINER_SAB, PNTS_SAB, RECRUIT_CNT, SAB_FLAG, RECRUIT_TRN_CNT, COUNT_TRN_FLAG, NO_SUP_SAB, RECRUIT_SUP_CNT, RECRUIT_MGR_CNT, COUNT_MGR_FLAG, PNTS_SAB_SUP, PREV_INV_SV, PREV_PSSV, IMAXSOFT13_SEQ_NO
FROM
SUNGB.DIST_G WHERE DIST_ID = :m

```
call      count      cpu    elapsed       disk      query    current    rows
------- ------- -------- ---------- ---------- ---------- ---------- -------
Parse         0     0.00       0.00          0          0          0       0
Execute 2098163    56.39      50.84          0          0          0       0
Fetch   4196326   153.42     155.87          0    8392703          0 2098163
------- ------- -------- ---------- ---------- ---------- ---------- -------
total   6294489   209.81     206.72          0    8392703          0 2098163

Misses in library cache during parse: 0
Optimizer goal: CHOOSE
Parsing user id: 31 (SUNGB)

Rows    Execution Plan
------- ---------------------------------------------------
      0 SELECT STATEMENT GOAL: CHOOSE
      0 TABLE ACCESS (BY INDEX ROWID) OF 'DIST_G'
      0 INDEX (UNIQUE SCAN) OF 'PK_DIST_G' (UNIQUE)
```

This query was executed 2098163 times during this batch job. This is a recurring theme that I'm going to be bringing up throughout this section since the number of times these queries are executed actually equal or far exceed the number of total rows in the tables! That is quite strange, and seems inefficient. So, this query was actually executed for every single row in the DIST_G table. However, the good news is that this query NEVER went to disk, and the total elapsed time for all executions of this query was 206 seconds (3 minutes 26 seconds).

You can see according to the execution plan (explain plan) that the query is indeed efficient, and is performing as it should.

SELECT IMAXSOFT13_SEQ_NO FROM SUNGB.AR_TRAN_G WHERE DIST_ID = :m ORDER BY IMAXSOFT13_SEQ_NO ASC

```
call      count      cpu    elapsed       disk      query    current    rows
------- ------- -------- ---------- ---------- ---------- ---------- -------
Parse         0     0.00       0.00          0          0          0       0
Execute  420286    15.66      14.51          0          0          0       0
Fetch    459658    26.43      26.01          0    1334390          0  129818
------- ------- -------- ---------- ---------- ---------- ---------- -------
total    879944    42.09      40.53          0    1334390          0  129818

Misses in library cache during parse: 0
Optimizer goal: CHOOSE
Parsing user id: 31 (SUNGB)

Rows    Execution Plan
------- ---------------------------------------------------
      0 SELECT STATEMENT GOAL: CHOOSE
      0 SORT (ORDER BY)
      0 TABLE ACCESS (BY INDEX ROWID) OF 'AR_TRAN_G'
      0 INDEX (RANGE SCAN) OF 'IDX_AR_TRAN_G' (NON-UNIQUE)
```

This query is actually executed almost 4x more than the actual number of records in the object! Again, the good news is that the total elapsed time of the execution/parse of these queries is only 40 seconds. The explain plan shows pure efficiency in the way the query is executed.

UPDATE SUNGB.DIST_G SET
DIST_ID=:v0, SPONSOR_ID=:v1, APPL_NAME=:v2, COUNTRY_CODE=:v3, ALLERGIC_FLAG=:v4, COUNT_FLAG=:v5, APPL_DT=:v6, STATUS=:v7, RANK=:v8,

```
GSV_ACCUM=:v9, GSV_ERR_ALLOW_FG=:v10, GSV_AMT_02=:v11, GSV_AMT_03=:v12,
GSV_AMT_04=:v13, GSV_AMT_05=:v14, GSV_AMT_06=:v15, GSV_AMT_07=:v16,
GSV_AMT_08=:v17, GSV_AMT_12=:v18, GSV_AMT_13=:v19, PSV_ACCUM=:v20,
PSV_ERR_ALLOW_FG=:v21, PSV_ERR_ALLOW_MO=:v22, PSV_AMT_02=:v23,
PSV_AMT_03=:v24, PSV_AMT_04=:v25, PSV_AMT_05=:v26, PSV_AMT_06=:v27,
PSV_AMT_07=:v28, PSV_AMT_08=:v29, PSV_AMT_12=:v30, PSV_AMT_13=:v31,
REBATE_AMT=:v32, MTD_1099_AMT=:v33, NUM_MO_INACTIVE=:v34,
GRP_DEV_BON_AMT=:v35, PRES_PNTS_MTD=:v36, NUM_OF_NEW_DIST=:v37,
PREV_AR_BAL=:v38, PREV_AR_BAL_CN=:v39, DIST_FLAG2=:v40, TRNS_SV=:v41,
INV_SV=:v42, PSV_BONUS=:v43, NUM_OF_NEW_ASST=:v44, NEW_TRAINER_SAB=:v45,
PNTS_SAB=:v46, RECRUIT_CNT=:v47, SAB_FLAG=:v48, RECRUIT_TRN_CNT=:v49,
COUNT_TRN_FLAG=:v50, NO_SUP_SAB=:v51, RECRUIT_SUP_CNT=:v52,
RECRUIT_MGR_CNT=:v53, COUNT_MGR_FLAG=:v54, PNTS_SAB_SUP=:v55,
PREV_INV_SV=:v56, PREV_PSSV=:v57
WHERE IMAXSOFT13_SEQ_NO = :m0
```

```
call      count       cpu    elapsed       disk      query    current    rows
------- ------- -------- ---------- ---------- ---------- ---------- -------
Parse         0     0.00       0.00          0          0          0       0
Execute 2460247   585.94     599.80          0    7397919    3109508 2460247
Fetch         0     0.00       0.00          0          0          0       0
------- ------- -------- ---------- ---------- ---------- ---------- -------
total   2460247   585.94     599.80          0    7397919    3109508 2460247

Misses in library cache during parse: 0
Optimizer goal: CHOOSE
Parsing user id: 31 (SUNGB)

Rows    Execution Plan
------- ---------------------------------------------------
      0 UPDATE STATEMENT GOAL: CHOOSE
      0 UPDATE OF 'DIST_G'
      0 INDEX (UNIQUE SCAN) OF 'I01_DIST_G' (UNIQUE)
```

This update was executed for each record in the database with a total elapsed time of 600 seconds (10 minutes). **Again, efficient access paths were used for this query**.

There are a few more queries that you can see in the output file, and it is really more of the same of what I have already mentioned.

Here is the aggregate information for ALL SQL performed by the batch job:

OVERALL TOTALS FOR ALL NON-RECURSIVE STATEMENTS

```
call      count       cpu    elapsed       disk      query    current    rows
------- -------- -------- ---------- ---------- ---------- ---------- -------
Parse         0     0.00       0.00          0          0          0       0
Execute  5547713   679.00     683.74          0    7397919    3109508 2460247
Fetch    5483705   225.69     227.88          0   12347884          0 3194557
------- -------- -------- ---------- ---------- ---------- ---------- -------
total   11031418   904.69     911.62          0   19745803    3109508 5654804

Misses in library cache during parse: 0
OVERALL TOTALS FOR ALL RECURSIVE STATEMENTS

Call     count       cpu    elapsed       disk      query    current    rows
------- ------ -------- ---------- ---------- ---------- ---------- -------
Parse     7636      0.60       0.55          0          0          0       0
Execute   7636      0.31       0.30          0          0          0       0
Fetch    11484      0.58       0.43          0      22908          0    7636
------- ------ -------- ---------- ---------- ---------- ---------- -------
total    26756      1.49       1.29          0      22908          0    7636

Misses in library cache during parse: 2
```

The total elapsed time in the database for the entire batch job was 913 seconds (15 minutes 13 seconds). What I would like to point out is the sheer volume of queries that are being

submitted by this batch job. **Essentially, the database is processing 11,031,418 queries in just over 15 minutes.** That is an **OUTSTANDING** measure of database performance. **Granted, NONE of the queries ever had to go to disk since all of the data easily fits into the SGA, but this is still an excellent measure of the performance of the database.** What can also be derived is that the application obviously has to parse a query, submit it, get the response, analyze the results, and then submit the next query. As I have mentioned before, this is what I feel is the problem. The batch job is the one that is trying to keep up with database, not the other way around. That also explains the amount of resource utilization we are seeing from the batch job, and the low amount of resource utilization we are seeing from the database background processes.

Oracle Statspack Samples Taken During Batch Job Run

The following excerpts were parsed out of the Statspack report that took database statistical snapshots in 15 minute intervals. The report covers the period during the batch job run. The below excerpt depicts the top 5 events in the database based upon elapsed time spent doing it. It is an indicator of serious wait events. Serious wait events almost certainly appear above the CPU time event. **However, you can see that the database is actually doing true work 85% of the time, which is excellent.** Any time spent waiting is on the redo log files, but with the amount of DML occurring within the batch job, this result is of no surprise.

```
Top 5 Timed Events
~~~~~~~~~~~~~~~~~~
Total
Event                                               Waits    Time (s) Ela Time (%)
--------------------------------------------- ------------ ----------- ------------
CPU time                                                        1,353       84.85%
log file parallel write                           22,155          117        7.32%
log file sequential read                           3,814           74        4.61%
control file sequential read                      12,108           33        2.04%
log file sync                                      3,676            8         .49%
```

The below excerpt shows that the database is getting all of its data out of the SGA (memory). There was only 1 disk read access the entire time. Tons of writes, but again, this is expected based upon how the batch job is architected. Also note the lack of waits for buffer resources, so the queries are NEVER waiting for data. As soon as the query is executed the data is returned. No waits at all.

```
Buffer Pool Statistics for DB: SUN Instance: SUN1 Snaps: 3 -11
-> Standard block size Pools D: default, K: keep, R: recycle
-> Default Pools for other block sizes: 2k, 4k, 8k, 16k, 32k
                                                          Free    Write Buffer
Number of  Cache      Buffer    Physical   Physical  Buffer Complete   Busy
Buffers    Hit %        Gets       Reads     Writes   Waits    Waits  Waits
---------- ----- ----------- ----------- ---------- ------- -------- ------
612,160    100.0 22,618,152           1    498,379       0        0      0
```

The below excerpt shows an unbelievable amount of activity for the UNDO segments. The good news here is that the process is never waiting for UNDO, so there is no problems in this area.

```
Rollback Segment Stats for DB: SUN Instance: SUN1 Snaps: 3 -11
->A high value for "Pct Waits" suggests more rollback segments may be required
->RBS stats may not be accurate between begin and end snaps when using Auto Undo
managment, as RBS may be dynamically created and dropped as needed

        Trans Table       Pct   Undo Bytes
RBS No      Gets        Waits     Written        Wraps  Shrinks  Extends
------ ------------ ------- --------------- -------- -------- --------
     0          32.0    0.00               0        0        0        0
     1      41,332.0    0.00     253,551,272      200        0        0
     2      30,981.0    0.00     154,326,188      114        0        1
     3      48,270.0    0.00     296,233,638      380        0       20
     4      30,603.0    0.00     173,984,436      118        0        1
     5      24,173.0    0.00      43,615,108        5        0        0
     6      27,053.0    0.00      64,748,460       31        1        0
```

7	33,393.0	0.00	190,614,584	200	0	3
8	39,880.0	0.00	222,820,278	127	0	0
9	53,227.0	0.00	342,071,966	196	0	35
10	24,530.0	0.00	45,979,200	45	0	0

The below excerpt shows that RAC functionality is not even part of the equation here since the batch job is only running on a single node. Therefore, RAC is not a part of the problem.

```
Cluster Statistics for DB: SUN Instance: SUN1 Snaps: 3 -11

Global Cache Service - Workload Characteristics
-----------------------------------------------
Ave global cache get time (ms):                      0.5
Ave global cache convert time (ms):                  0.0

Ave build time for CR block (ms):                    0.0
Ave flush time for CR block (ms):                    0.0
Ave send time for CR block (ms):                     0.1
Ave time to process CR block request (ms):           0.1
Ave receive time for CR block (ms):

Ave pin time for current block (ms):                 0.0
Ave flush time for current block (ms):               0.0
Ave send time for current block (ms):                0.0
Ave time to process current block request (ms):      0.0
Ave receive time for current block (ms):

Global cache hit ratio:                              0.0
Ratio of current block defers:                       0.0
% of messages sent for buffer gets:                  0.0
% of remote buffer gets:                             0.0
Ratio of I/O for coherence:                          0.0
Ratio of local vs remote work:
Ratio of fusion vs physical writes:                  0.0

Global Enqueue Service Statistics
---------------------------------
Ave global lock get time (ms):                       0.0
Ave global lock convert time (ms):                   0.0
Ratio of global lock gets vs global lock releases:   1.0

GCS and GES Messaging statistics
--------------------------------
Ave message sent queue time (ms):                    0.1
Ave message sent queue time on ksxp (ms):            0.5
Ave message received queue time (ms):                0.0
Ave GCS message process time (ms):                   0.3
Ave GES message process time (ms):                   0.2
% of direct sent messages:                          97.4
% of indirect sent messages:                         0.5
% of flow controlled messages:                       2.0
```

There are plenty of more statistics in the report, but I wanted to only pick out the sections that needed mentioning. If I didn't mention something, that means that the metric is either in satisfactory levels, or just is not existent (in other words there are NO waits or contentions at all, or anything else in the database that would indicate a problem with the database not performing appropriately).

Conclusions

1. The database instance is performing **at peak performance**.
2. The queries submitted by the batch job are written **extremely efficient**, and thanks to the proper use of bind variables, the throughput of servicing the queries is not affected.
3. Since RAC functionality is not even being used in the environment during the batch job, it can not be presented as a culprit for database performance issues.

So what is the problem? The sheer number of query executions (>11 million) performed by the batch job is what is causing the durations we are currently seeing. Again, think about what needs to happen:

1. Batch job parses and submits a query
2. Database executes the query (unless it must parse first which it almost never does)
3. Data set is created
4. Data set is sent back to client
5. Data set is analyzed and processed
6. Batch job parses and submits next query

I have already shown that the database is spending a total of 15 minutes doing the query work. The other 1 hour and 45 minutes is being spent performing steps 4-6. Network isn't an issue in this case since the batch job is running locally on the database server. The hang-up is within the control of the application.

Recommendations

1. Efficiencies need to be added to the application code to try to cut down on the number of recursive query executions and try to satisfy more records with fewer queries. In other words, could the app be rewritten to perform updates on more than one record at a time? Instead of a separate UPDATE for each record, have one UPDATE to handle 100 records (or whatever makes sense). For the SELECTs, why select each record one at a time? You could potentially handle this with the use of cursors where all viable records are selected at one time into a structure that you can fetch from by popping records off the stack one at a time until you process the whole stack. Using a cursor requires the use of only one SELECT statement instead of millions of SELECT statements.
2. Make use of your RAC architecture! Try to figure out how to parallelize the batch job to possibly do half the processing on one node, and the other half on the other. This would require that the job be moved off the database server to be run on an app server.