Accelerate Oracle Logs and Tablespaces Using the Write Accelerator

Mike Ault

TMS
TEXAS MEMORY SYSTEMS

September 2008

Contents

Section 1

Executive Summary

Texas Memory Systems is now offering the Write Accelerator product for databases. The Write Accelerator offers relief from slowdowns caused by – as its name implies – waits for writes. In the Oracle universe data writes are not usually an issue as Oracle uses the delayed block cleanout process that does what is known as lazy-writes, only writing dirty blocks back to disk when absolutely needed. However, there are three aspects of Oracle writes that need write speeds to be as fast as possible: writes to the redo logs, writes to the temporary tablespace, and writes to the undo tablespace. Let's look at writes to the redo logs first.

Section 2

Redo Log Writes

With the performance gap between processors and hard drive-based storage systems widening, solid state storage is entering the limelight. Because solid state systems rely on memory chips for data storage, they offer unprecedented access times that narrow the gap between the processor speeds and storage speeds. Companies have used solid state storage systems to resolve input/output (I/O) performance problems for over three decades. These systems have become increasingly sophisticated, higher performing, and lower cost, which sends a clear message … there is no better tool for improving I/O performance.

The reality of the Oracle database universe is that databases and transactions are getting larger, more numerous, and more complex each day. As transactions grow more numerous and longer in duration the redo logs see a considerable increase in stress levels. This increase in stress shows up as redo log related wait events in the Oracle wait interface. In order to alleviate this stress DBAs usually end up moving redo logs to separate file systems, sometimes even separate disk RAID areas.

Redo logs are also getting larger as the number of concurrent transactions they service increase in number and size. The DBA must balance the needed write time for a very large log against the penalties of numerous small redo log writes if the log is undersized. Redo log sizes of several hundred megabytes are common on large transactional databases; logs of this size require a discrete amount of time to be written to traditional mechanical disk assets, whether the logs are interspersed across the database RAID set or have their own dedicated disk array.

Redo log writes are traditionally triggered at buffers being 1/3rd full or when the redo log reaches 1 megabyte in size. As you can see this occurs quite quickly with the 1 megabyte size limit if you have a redo log that is several hundred megabytes in size. One parameter available in Statspack and AWR reports that tells if your redo logs are oversized is the redo wastage statistic. If this value is large in ratio to the number of redo write events then your logs are probably oversized.

Log file sync, redo write time, log file parallel write: these are all wait events concerning the LGWR (Oracle log writer) process. Do high values for these waits mean that Oracle is stressed as far as the writing of redo writes to the redo logs? The answer is a definite maybe. A part of a system cannot be taken in isolation; OracleLGWR is a part of the Oracle system which itself is a part of the overall computer environment. Anything in the computer environment or anything in the Oracle system that is stressing the IO subsystem can result in stress for the LGWR system. The problem may be as easily solved as (on Linux or UNIX) renicing the LGWR process to have a higher priority over the database writer process (DBWR) or by eliminating non-needed monitoring processes. Also, an important thing to

remember is to use Statspack or AWR tables or reports to isolate the time period of concern with any statistics, as straight from the V$ dynamic performance views they are cumulative values since the time the database started.

However, just because you have log file sync wait events doesn't mean that you have IO issues with redo logs. When a process posts data to the log buffer and signals LGWR to write it goes into a sleep state and thus starts a log file sync event. Now, if LGWR is just sitting idle then it will immediately wake up and write the log information, thus the wait will be of short duration. However, if LGWR is busy doing other writes then the new log write has to wait. Since LGWR is a user mode process, this type of activity will require transitions to kernel mode and back again. During these transitions LGWR can lose the CPU to processes with higher priorities, thus the suggestion about renicing the LGWR process. If LGWR uses the CPU it goes into a spin mode and this is all part of the log file sync wait event timing. All of this CPU wait time, which has nothing to do with IO at all, is added into the log file sync event timing.

Far more telling for the redo logs are the values for the actual write times: redo write time and log file parallel writes. Of course, as with log file syncs, be sure that you isolate these statistics for the time period of concern using Statspack or AWR reports or tables. By using the statistic Redo Log Writes you can find out just what each redo write or log parallel write is costing you per write. Once you have the time per write you can truly determine if the IO subsystem is causing your issues.

Texas Memory System's StatspackAnalyzer.com Web site analyzes many Statspack and AWR reports daily. Some clients also send in their reports for more detailed analysis. An example top five waits section of a typical report that seems to indicate redo log related issues is shown in Figure 1.

```
Top 5 Timed Events
~~~~~~~~~~~~~~~~~~                                          % Total
Event                            Waits        Time (s)      Ela Time
-------------------------- ------------- -------------- ---------------
log file sync              21,470,106        125,416           34.86
CPU time                                      68,364           19.00
db file sequential read    35,725,749         46,269           12.86
db file scattered read     10,201,389         45,346           12.60
log file parallel write    11,649,482         43,993           12.23
Replication Dequeue         3,534,856         19,297            5.36
```

Figure 1: Statspack Showing Redo Log Stress

Looking at Figure 1 we see that both log file sync and log file parallel write are showing up in the top five events, with log file sync being the top event. Figure 2 shows the background wait events for the same statspack report.

```
Background Wait Events for DB: (obscured)  Instance: heart  Snaps: 2237 -
2238
          -> ordered by wait time desc, waits desc (idle events last).
                                                          Avg
                                                Total Wait  wait   Waits
Event                          Waits   Timeouts   Time (s)  (ms)   /txn
---------------------------- ------- ---------- -------- -------- -------
log file parallel write   11,649,477       0     43,993       4      0.5
log file sequential read     100,458       0        565       6      0.0
control file parallel write   31,171       0         67       2      0.0
LGWR wait for redo copy      610,215      59         34       0      0.0
latch free                     5,753   1,219         19       3      0.0
db file single write          10,890       0          8       1      0.0
db file sequential read       13,869       0          5       0      0.0
log file single write            400       0          4       9      0.0
```

Figure 2: Background Wait Events

Looking deeper into the report we see that the number of redo writes is 11,649,440 and the total redo write time is 12,418 seconds yielding 1 millisecond per write, indicating a good portion of our wait time is being taken up with other non-IO operations.

So, what would the proper course of action be to correct this redo log situation? First, we would need to find out a few more things from the DBA for the system. For example, what are the run queue numbers for the time period in question? Are the redo logs on separate devices? Has the LGWR process been reniced? Now, if the answers are that there is no appreciable run queue (or the run queue is due to IO waits), the redo logs have been placed optimally, and the LGWR has been reniced and they are still seeing this wait event profile, then more advanced means of controlling redo log waits is indicated.

We don't, however, have an "after" report for this Statspack report where the logs have been moved to a TMS Write Accelerator. Let's force redo writes in an application; the code listed in Figure 3 runs against a table with 1 billion rows so we should generate a decent amount of redo when it is run.

```
CREATE OR REPLACE PROCEDURE force_undo(num_rec NUMBER) AS
CURSOR get_records IS SELECT * FROM h_lineitem WHERE rownum<numrec;
line_record  h_lineitem%ROWTYPE;
i INTEGER;
BEGIN
        i:=1;
        FOR get_records_rec IN get_records LOOP
                INSERT INTO dup_lineitem VALUES get_records_rec;
                i:=i+1;
                IF i=10 THEN
                        COMMIT;
                i:=1;
                END IF;
        END LOOP;
END;
/
```

Figure 3: Redo Generating Script

For expediency sake the test was limited to the first 48,220,000 records in the referenced table.

For our test we will use two systems; the first is a 24-72 GB 10K JBOD based system using ASM to manage the disks in a stripe and mirror everything (SAME) configuration. The second is a system using a TMS Write Accelerator-based technology for storage.

Results from the standard disk based system (some values summed across 2 instances) are shown in Figure 4.

```
           Snap Id         Snap Time     Sessions   Curs/Sess
          ---------- ------------------ -------- -----------
Begin Snap:   144     20-Aug-08 16:23:46      41        1.5
  End Snap:   147     20-Aug-08 18:43:26      40        1.3
   Elapsed:             139.68 (mins)
   DB Time:             173.88 (mins)

Cache Sizes                Begin     End
~~~~~~~~~~~              ---------- ----------
       Buffer Cache:      2,360M    2,360M      Std Block Size       8K
       Shared Pool Size:    512M      512M      Log Buffer:     10,604K

Load Profile         Per Second    Per Transaction    Per Exec   Per Call
~~~~~~~~~~~~         -----------   ----------------- ----------- ---------
       Redo size:    2,588,920.9       4,047.5
   Logical reads:        8,023.1          12.5
   Block changes:       13,368.5          20.9
  Physical reads:          106.7           0.2
 Physical writes:          163.5           0.3
       Executes:         5,757.5           9.0
    Transactions:           639.6

Top 5 Timed Foreground Events
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                                                    Avg
                                                   Wait     %DB
Event                            Waits  Time(s)  (ms)    time   Wait Class
------------------------------ ------- -------- ------ ------- ----------
DB CPU                                    7,891         81.7
PX qref latch                28,408,603     399     0     4.1        Other
log file switch (checkpoint in    22,219     385    17     4.0   Configurat
PX Deq: reap credit            3,621,380     205     0     2.1        Other
log file switch completion         2,239     122    55     1.3   Configurat
Foreground Wait Events                DB/Inst: AULTDB/aultdb1  Snaps: 144-
147
-> s  - second, ms - millisecond -    1000th of a second
-> Only events with Total Wait Time (s) >= .001 are shown
-> ordered by wait time desc, waits desc (idle events last)
-> %Timeouts: value of 0 indicates value was < .5%.  Value of null is
truly 0
                                                          Avg
                                        %Time Total Wait  wait   Waits    % DB
Event                            Waits  -outs   Time (s) (ms)    /txn    time
-------------------------- ------------ ----- ---------- ------- --------
log file switch (checkpoint   22,219        0        385    17     0.0     4.0
log file switch completion     2,239        0        122    55     0.0     1.3
log file sync                     18        0          0    13     0.0      .0
          ---------------------------------------------------------------
Background Wait Events                DB/Inst: AULTDB/aultdb1  Snaps: 144-
147
-> ordered by wait time desc, waits desc (idle events last)
-> Only events with Total Wait Time (s) >= .001 are shown
-> %Timeouts: value of 0 indicates value was < .5%.  Value of null is
truly 0
```

```
                                                          Avg
                                    %Time Total Wait     wait   Waits      % bg
Event                       Waits -outs    Time (s)    (ms)    /txn      time
-------------------------- ------ ----- ---------- ------- -------- ------
log file parallel write  405,691     0      7,247      18     0.1      71.4
log file single write      1,908     0         23      12     0.0        .2
log file sequential read   1,908     0         15       8     0.0        .1
log file switch (checkpoint  215     0          3      15     0.0        .0
latch: redo writing           10     0          0       1     0.0        .0
         ------------------------------------------------------------


SQL ordered by Elapsed Time           DB/Inst: AULTDB/aultdb1  Snaps: 144-
147
-> Resources reported for PL/SQL code includes the resources used by all
SQL
   statements called by the code.
-> % Total DB Time is the Elapsed Time of the SQL statement divided
   into the Total Database Time multiplied by 100
-> Total DB Time (s):           9,653
-> Captured SQL account for     151.1% of Total

  Elapsed      CPU                  Elap per  % Total
  Time (s)   Time (s)  Executions   Exec (s)  DB Time    SQL Id
---------- ---------- ------------ ---------- ------- -------------
     3,818      3,222   48,217,432        0.0    39.6 g1dppfmhtxs00
Module: SQL*Plus
INSERT INTO DUP_LINEITEM VALUES (:B1 ,:B2 ,:B3 ,:B4 ,:B5 ,:B6 ,:B7 ,:B8
,:B9 ,:B10 ,:B11 ,:B12 ,:B13 ,:B14 ,:B15 ,:B16 )

Instance Activity Stats            DB/Inst: AULTDB/aultdb1  Snaps: 144-147

Statistic                            Total         per Second      per Trans
-------------------------------- ------------------ -------------- -------
redo blocks checksummed by FG (e  765,578              91.4          0.1
redo blocks read for recovery           0               0.0          0.0
redo blocks read total                  0                0.          0.0
redo blocks written            43,998,109            5,249.8          8.2
redo blocks written for direct w      296               0.0          0.0
redo buffer allocation retries      1,427               0.2          0.0
redo entries                   58,256,087            6,951.1         10.9
redo log space requests            25,142               3.0          0.0
redo log space wait time           51,083                6.          0.0
redo ordering marks               518,019              61.8          0.1
redo size                  21,697,424,708        2,588,920.9      4,047.5
redo subscn max counts             14,464               1.7          0.0
redo synch time                        38               0.0          0.0
redo synch writes                     169               0.0          0.0
redo wastage                  101,846,820           12,152.3         19.0
redo write time                   757,698              90.4          0.1
redo writer latching time             141               0.0          0.0
redo writes                       405,691              48.4          0.1
user commits                    5,359,847             639.5          1.0
```

Figure 4: AWR Data from Standard Disk Based System

The AWR results from the second TMS Write Accelerator based system are shown in Figure 5.

```
Snap Id   Snap Time            Sessions  Curs/Sess
            --------- ------------------- -------- ---------
Begin Snap:     615   20-Aug-08 15:21:45         18          4.1
  End Snap:     617   20-Aug-08 16:17:02         16          2.4
   Elapsed               .28 (mins)
   DB Time:           482.54 (mins)
```

```
Cache Sizes
                     Begin          End
                   ---------- ----------
  Buffer Cache:     2,336M       2,336M         Std Block Size:      8K
Shared Pool Size:     688M         688M             Log Buffer: 14,404K


Load Profile
~~~~~~~~~~~~      Per Second       Per Transaction
                                  ---------------       ---------------
                  Redo size:          6,437,151.88              3,984.87
              Logical reads:             21,439.02                 13.27
              Block changes:             33,887.42                 20.98
             Physical reads:                269.65                  0.17
            Physical writes:                408.30                  0.25
                  Executes:             14,540.05                  9.00
                Transactions:              1,615.40


Top 5 Timed Events                                    Avg     %Total
                                                     Wait       Call       Wait
Event                            Waits  Time (s)     (ms)       Time      Class
------------------------------ ------------ ----------- ------ ------ ----
------
PX Deq Credit: send blkd       3,669,884    25,566       7       88.3      Other
CPU time                                     2,594                9.0
log file parallel write        5,313,254      495       0        1.7 System I/O
log file switch (checkpoint in 1,121          407     363        1.4 Configurat
log file switch completion     1,392          404     290        1.4 Configurat
          -------------------------------------------------------------
Wait Events                        DB/Inst: TMSORCL/tmsorcl  Snaps: 615-617
-> s  - second
-> cs - centisecond -     100th of a second
-> ms - millisecond -    1000th of a second
-> us - microsecond - 1000000th of a second
-> ordered by wait time desc, waits desc (idle events last)

                                                                      Avg
                                        %Time   Total Wait      wait   Waits
Event                            Waits   -outs     Time (s)      (ms)   /txn
------------------------------ -------- --------- ------------ ------- -----
log file parallel write        5,313,254       .0          495             1.0
log file switch (checkpoint    1,121          32.6         407      363    0.0
log file switch completion     1,392          27.4         404      290    0.0
log file single write          1,082           .0           0        0    0.0
log file sync                     56           .0           0        1    0.0
LGWR wait for redo copy        3,402           .0           0        0    0.0
log file sequential read       1,082           .0           0        0    0.0
latch: redo allocation            59           .0           0        0    0.0
latch: redo writing                1           .0           0        0    0.0
log file parallel write        5,313,254       .0         495        0    1.0
log file switch (checkpoint        9           .0           2      229    0.0
log file single write          1,082           .0           0        0    0.0
log file sequential read       1,082           .0           0        0    0.0
log file sync                      2           .0           0        0    0.0
latch: redo writing                1           .0           0        0    0.0
          -------------------------------------------------------------


SQL ordered by Elapsed Time       DB/Inst: TMSORCL/tmsorcl  Snaps: 615-617
-> Resources reported for PL/SQL code includes the resources used by all
SQL statements called by the code.
-> % Total DB Time is the Elapsed Time of the SQL statement divided
   into the Total Database Time multiplied by 100


  Elapsed      CPU                  Elap per  % Total
  Time (s)   Time (s)  Executions   Exec (s)  DB Time    SQL Id
---------- ---------- ------------ ---------- ------- -------------
     1,805      1,051   48,219,882        0.0     6.2 g1dppfmhtxs00
Module: SQL*Plus
INSERT INTO DUP_LINEITEM VALUES (:B1 ,:B2 ,:B3 ,:B4 ,:B5 ,:B6 ,:B7 ,:B8
,:B9 ,:B10 ,:B11 ,:B12 ,:B13 ,:B14 ,:B15 ,:B16 )
```

Instance Activity Stats DB/Inst: TMSORCL/tmsorcl Snaps: 615-617

Statistic	Total	per Second	per Trans
redo blocks read for recovery	0	0.0	0.0
redo blocks written	45,944,613	13,851.8	8.6
redo buffer allocation retries	1,416	0.4	0.0
redo entries	35,871,499	10,814.9	6.7
redo log space requests	3,051	0.9	0.0
redo log space wait time	83,113	25.1	0.0
redo ordering marks	271,248	81.8	0.1
redo size	21,351,144,464	6,437,151.9	3,984.9
redo synch time	7	0.0	0.0
redo synch writes	929	0.3	0.0
redo wastage	1,427,070,848	430,247.3	266.3
redo write time	55,447	16.7	0.0
redo writer latching time	8	0.0	0.0
redo writes	5,313,254	1,601.9	1.0
user commits	5,358,050	1,615.4	1.0

Figure 5: TMS Write Accelerator based system results

For the disk based system the time to complete a redo write was 757,698 centiseconds/405,691 writes for a value of 18.7 milliseconds per write. From the above excerpts we can calculate that the time per redo write with the TMS Write Accelerator was 55,447 centiseconds/5,313,254 writes or 0.104 milliseconds per write.

From a direct ratio of the write times, the TMS Write Accelerator increased the write speed by a factor of over 13 times for the redo log writes alone.

When all tuning efforts result in unsatisfactory redo log write performance, obviously the only course of action is to reduce latency for the write process. The only way to reduce latency to its absolute minimum is to put your redo logs on low latency storage such as the TMS Write Accelerator.

Section 3

Temporary Tablespace Writes and Reads

Temporary tablespaces in Oracle have been a thorn in the DBAs side since the very beginning. A necessary evil, temporary tablespaces are the swap areas of Oracle. If a process requires more data than can be held in memory, then temporary space is used. Of course the only processes that usually require large quantities of data are sorts, hash joins, and large temporary table operations; a sub-set of sorts is the bitmap operation which can also overflow to the temporary tablespace.

Of all of the operations mentioned above, the only ones that are adequately monitored via easily obtainable statistics are sorts. In the Statspack and AWR reports there are sort statistics and PGA histograms showing how sort activity was divvied up to different sort size categories. Unfortunately hash, global temporary table operations, and bitmap operations have no such statistics, histograms, or other means of being tracked.

Hash and bitmap operations can be crudely tracked using the V$SQL_PLAN table, but only based on current SQL operations. Inflight operations for hash segments can also be tracked in the V$SORT_USAGE table, but only for sorts and hash operations that are currently running.

The only way to reliably see if there is excessive temporary usage is to monitor the IO to the temporary tablespace. I have seen cases where there were no sorts to disk indicated by the sorts (disk) parameter and yet the IO to the temporary tablespace was the top amount of IO in the entire database system. This is shown from an actual client Statspack excerpt in Figure 6.

```
Tablespace IO Stats for DB: <obfc>  Instance: <obfc>1  Snaps: 1964 -1967
->ordered by IOs (Reads + Writes) desc

Tablespace
------------------------------
                     Av        Av        Av        Av           Buffer  Av Buf
           Reads Reads/s    Rd(ms)   Blks/Rd    Writes  Writes/s   Waits  Wt(ms)
-------------- ------- ------ ------- ------------ -------- ---------- ---
INVD1
         715,082     795      0.7       7.8        24         0       0     0.0
ONTD1
          68,910      77      7.8       1.0         6         0       0     0.0
TEMP02
          26,339      29     24.6       7.0    30,012        33       0     0.0
WSHD
          48,377      54      8.6       6.9         5         0       0     0.0
ARD1
          19,954      22      6.1       3.6        16         0       0     0.0
POD
           ------------------------------------------------------------
```

Figure 6: Tablespace IO Section of Statspack

```
Date: 11/30/07                                                    Page:  1
Time: 08:24 AM              IO Timing Analysis                    PERFSTAT
                             <Obfc> database

InstanceFILE#File Name                     PHYRDS PHYWRTS  PHY_RD_TM PH_WRT_TIM
-------- ---------- ------------------------------ ------- -------- ------
 1       10  ALRX /dbpreq01/temp02_004.dbf  32670  411231 15.7459137 3.21591271
 1        8  AKX /dbpreq01/temp02_002.dbf  155741  404184 14.5146943 6.27315777
 1        9  ALRD /dbpreq01/temp02_003.dbf 103063  406668 14.3672317 3.60064475
 1        7  AKD /dbpreq01/temp02_001.dbf  194301  419025 12.9342463 2.58877394
 1       12  AMSX /dbpreq01/temp02_006.dbf  46679  415731 10.2241265 4.47436924
 1       11  AMSD /dbpreq01/temp02_005.dbf 400527  611838 7.22214732 10.6046323
 4        7  AKD /dbpreq01/temp02_001.dbf    1050    1925 1.518095241  7.1942857
 3        7  AKD /dbpreq01/temp02_001.dbf    1054    1332 .9117647067  .53978979
 2        7  AKD /dbpreq01/temp02_001.dbf     703     981 .739687055 6.59734964
```

Figure 7: IO Timings Report

As you can see, the write timings vary from a low of 2.5 ms to over 17 ms. This report segment is also disturbing because of the extremely long read times shown in concert with the write times. Moving the temporary segments for this database to the Write Accelerator would not only drop write times to less than a millisecond, it would also drop read times to that level, providing immediate, permanent relief from temporary tablespace IO stress.

If the temporary tablespace is not isolated to its own physical area of the disk array, its IO operations can significantly degrade the overall performance of the array if temporary tablespace IO becomes the major source of IO operations in the database. The temporary tablespace should be placed in an "isolated" area from the other tablespace datafiles to prevent its IO operations from degrading their performance.

For example, the following AWR excerpts are from two systems running a sort against a 1 billion row table (SELECT * FROM h_lineitem ORDER BY l_comment,l_shipinstruct). The database, using a 24 disk ASM managed JBOD array, took 409.14 minutes (6 hours 49 minutes) to complete the task. A system using Write Accelerator technology took 41.39 minutes.

Figure 8 shows the AWR results from a system using 24-72 GB 10K Seagate hard drives with management by Oracle ASM.

```
     Service Statistics
     Service Name
     ------------------------------------------------------------------
  User I/O  User I/O   Concurcy  Concurcy      Admin     Admin   Network   Network
 Total Wts    WtTime  Total Wts    WtTime  Total Wts  Wt Time Total Wts  Wt Time
 --------- --------- --------- --------- --------- --------- --------- ------
 SYS$USERS
   2160604      1763       348       189         0         0   7206979     1350
          ------------------------------------------------------------------

   SQL Statistics
     Elapsed      CPU                 Elap per  % Total
     Time (s)   Time (s)  Executions   Exec (s)  DB Time    SQL Id
   ---------- ---------- ------------ ---------- ------- -------------
       40,686      8,506            1    40686.3    92.5 1avpjg9pcsgnb
   Module: SQL*Plus
   select * from h_lineitem order by l_comment,l_shipinstruct
```

```
Tablespace IO Statistics
Tablespace
------------------------------
               Av        Av       Av                   Av    Buffer  Av Buf
   Reads   Reads/s    Rd(ms)  Blks/Rd    Writes   Writes/s     Waits  Wt(ms)
-------- --------- -------- --------- -------- ----------- ------- ------
TEMP
1,985,044      81      2.1       1.0   77,761           3         0    0.0
          -------------------------------------------------------------

Snap 124 to 132 IO Timing Analysis

FILENAME                       PHYRDS     PHYWRTS   AVE_RD_TIM    AVE_WRT_TIM
------------------------------ ---------- ---------- ---------- ---------
+DATA2/temp.274.660989059    706,936      27,691          2.68           7.78
+DATA2/temp.273.660930207    594,287      23,282          1.63           7.54
+DATA2/temp.266.660756117    683,821      26,788          1.82           7.36
```

Figure 8: Temporary write data from standard disk based system

Figure 9 shows the same test, only run against a system using a single TMS Write Accelerator solid state drive (SSD).

```
Service Statistics
Service Name
----------------------------------------------------------------

 User I/O   User I/O   Concurcy  Concurcy      Admin    Admin    Network  Network
Total Wts    Wt Time  Total Wts  Wt Time  Total Wts  Wt Time  Total Wts  Wt Time
--------- --------- --------- --------- --------- --------- --------- ----
SYS$USERS
  2001299         76        132         1          0         0    7200346       40
        ------------------------------------------------------------

SQL Statistics
  Elapsed      CPU                 Elap per  % Total
  Time (s)   Time (s)  Executions   Exec (s)  DB Time    SQL Id
---------- ---------- ---------- ---------- ------- -------------
    12,469      1,390          1    12469.1    99.4 1avpjg9pcsgnb
Module: SQL*Plus
select * from h_lineitem order by l_comment,l_shipinstruct

Tablespace IO Statistics
Tablespace
------------------------------
                    Av       Av      Av                Av  Buffer    Av Buf
        Reads  Reads/s   Rd(ms) Blks/Rd   Writes Writes/s  Waits    Wt(ms)
------------- --------- ------ --------- ------- --------- ------- ------
TEMP
    1,981,420      798      0.1     1.0  115,057        46     342       5.6
          ------------------------------------------------------------

Snap 585 to 586 IO Timing Analysis
FILENAME                PHYRDS    PHYWRTS AVE_RD_TIM AVE_WRT_TIM
------------------- ---------- ---------- ---------- -----------
/ssd1/temp101.dbf       1721221      98681       0.08        1.75
/ssd2/temp102.dbf        260199      16376       0.09        1.71
```

Figure 9: Temporary Write Data to a TMS Write Accelerator Based System

Even with write caching enabled (as shown by the 1-2 millisecond writes on the temporary tablespace in the non-Write Accelerator system), the Write Accelerator technology beats the execution of the statement by a factor of 4 or more. So from placing temporary write data on Write Accelerator technology we reduced the execution time for the statement by over 400%.

So once you optimally tune the sort_area_size, hash_area_size, bitmap_create_area_size, and bitmap_merge_area_size parameters or find a proper setting for the pga_aggregate_target parameter and you are still beating the temporary tablespace to death, then it is time to consider a Write Accelerator.

Generally temporary tablespace reads and writes will be done in direct mode, so the direct read and direct write statistics will show the volume of reads and writes to the temporary tablespace. If you are experiencing extreme reads and writes to your temporary tablespace, the write accelerator will definitely improve your IO timing profile and application performance.

Section 4

Undo Tablespace Writes

Undo writes occur when a transaction changes data. Undo segments are generated to hold before image records of data that is being changed. If for some reason the transaction is aborted, or the user deliberately rolls back the transaction, the undo data is used to restore the block data to current mode as it was before the change began. In a high manual transaction environment undo tablespace reads and writes can become a major source of IO in the system, especially if IO waits are a concern.

In order to demonstrate this we create an artificial situation where a process reads a record from table a, writes it to table b, then rolls back every thousand rows. The table has 56 million rows so this process will generate a large undo load. The script for this is shown in Figure 10.

```
   CREATE OR REPLACE PROCEDURE force_undo AS
   CURSOR get_records IS SELECT * FROM h_lineitem;
   line_record  h_lineitem%ROWTYPE;
   i INTEGER;
   BEGIN
      i:=1;
      FOR get_records_rec IN get_records LOOP
         INSERT INTO dup_lineitem VALUES get_records_rec;
         i:=i+1;
      IF i=1000 THEN
         ROLLBACK;
         i:=1;
      END IF;
   END LOOP;
END;
/
```

Figure 10: Script to generate undo segment activity

The load profile for a disk based system from AWR is shown in Figure 11.

```
Load Profile             PerSecond    PerTransaction       Per Exec     Per Call
~~~~~~~~~~~~         --------------    --------------- ---------- -------
      DB Time(s):              2.4              0.6           0.00         5.19
       DB CPU(s):              0.9              0.2           0.00         2.07
       Redo size:  2,285,108.3        536,801.8
   Logical reads:     15,421.9          3,622.8
   Block changes:     16,238.6          3,814.7
  Physical reads:         74.5             17.5
 Physical writes:         38.1              9.0
      User calls:          0.5              0.1
          Parses:          3.0              0.7
     Hard parses:          0.1              0.0
W/A MB processed:    130,328.7         30,615.9
          Logons:          0.0              0.0
        Executes:      3,907.7            918.0
       Rollbacks:          4.0              0.9
    Transactions:          4.3
```

Figure 11: Undo Activity for Standard DiskBbased System

As you can see the rollbacks/sec and transactions per second are both at around 4 per second with 3,908 executions per second from a single process. It is hoped that no real database would generate this type of undo load. Instead, look for about 1 rollback per 1000 insert executions, which is exactly what we should be seeing from our procedure.

This large number of transactions and rollbacks places a medium load on the CPU, as is shown in the next AWR extract in Figure 12.

```
Host CPU (CPUs:    2 Cores:    1 Sockets:    1)
~~~~~~~~              Load Average
                   Begin       End     %User   %System      %WIO     %Idle
               --------- --------- --------- --------- --------- ---------
                    0.40      0.40      32.1      20.1       3.4      46.5

Instance CPU
~~~~~~~~~~~~
                % of total CPU for Instance:  48.0
                % of busy  CPU for Instance:  89.7
  %DB time waiting for CPU - Resource Mgr:   0.0
```

Figure 12: CPU Load for standard disk system

As you can see we have almost no %WIO (percent wait for IO), only 3.4%. Everything at this point is CPU driven.

However, looking at the events section of the report in Figure 13 shows that all the wait events slowing the system down are IO driven events.

```
                                                          Avg
                                        %Time Total Wait  wait   Waits   % DB
Event                            Waits  -outs   Time (s)  (ms)   /txn   time
-------------------------- ------------ ----- ---------- ------- -------
log file switch completion      2,089      0        212     102     0.0     .7
db file sequential read        11,077      0         82       7     0.2     .3
log file sequential read          445      0          7      15     0.0     .0
control file sequential re        827      0          5       7     0.0     .0
direct path read                7,280      0          4       1     0.1     .0
-------------------------------------------------------------
```

Figure 13: Events in a Standard Disk System with High Undo Activity

The main transaction we are seeing is of course our INSERT transaction; there should be one for every row in our source table as shown in Figure 14.

```
  Elapsed      CPU                  Elap per  % Total
  Time (s)   Time (s)  Executions   Exec (s)  DB Time    SQL Id
---------- ---------- ------------ ---------- ------- -------------
     2,578      2,416   48,202,499        0.0     8.9 g1dppfmhtxs00
Module: SQL*Plus
INSERT INTO DUP_LINEITEM VALUES (:B1 ,:B2 ,:B3 ,:B4 ,:B5 ,:B6 ,:B7 ,:B8
,:B9 ,:B10 ,:B11 ,:B12 ,:B13 ,:B14 ,:B15 ,:B16 )
```

Figure 14: SQL Executions In an Hour on the Disk Based System

So we can see that as with the SSD based system we processed at least 48,202,000 rows from our source table.

The main transaction-related statistics that show undo activity are:

- DBWR undo block writes – number of times dbwr wrote undo blocks
- Transaction rollbacks – count of rollback commands
- Data blocks consistent reads – undo records applied
- Rollback changes – undo records applied
- Undo change vector size – cumulative size in bytes of undo changes.

Figure 15 shows some of the events relating to our transactions on the disk based system.

```
Instance Activity Stats            DB/Inst: AULTDB/aultdb2  Snaps: 208-211

Statistic                                 Total   per Second      per Trans
-------------------------------- ------------------ ------------ -------
DBWR undo block writes                    437,496         35.4           8.3
Effective IO time                      46,072,596      3,728.3         875.8
commit batch/immediate performed           48,268          3.9           0.9
commit batch/immediate requested           48,268          3.9           0.9
commit immediate performed                 48,268          3.9           0.9
commit immediate requested                 48,268          3.9           0.9
concurrency wait time                       3,717          0.3           0.1
data blocks consistent reads - u                7          0.0           0.0
enqueue timeouts                               14          0.0           0.0
enqueue waits                                  17          0.0           0.0
execute count                          48,290,141      3,907.7         918.0
opened cursors cumulative              48,335,614      3,911.4         918.8
physical read IO requests                  29,925          2.4           0.6
physical read bytes                 7,538,368,512    610,018.7     143,301.4
physical read total IO requests            77,659          6.3           1.5
physical read total bytes           8,713,876,480    705,143.0     165,647.3
physical read total multi block             7,489          0.6           0.1
physical reads                            920,211         74.5          17.5
physical reads cache                       21,925          1.8           0.4
physical reads cache prefetch               7,177          0.6           0.1
physical reads direct                     898,286         72.7          17.1
physical reads direct temporary             1,895          0.2           0.0
physical reads prefetch warmup                449          0.0           0.0
physical write IO requests                 55,617          4.5           1.1
physical write bytes                3,857,481,728    312,154.6      73,329.2
physical write total IO requests           225,37          18.2           4.3
physical write total bytes         62,774,493,184  5,079,828.0   1,193,318.0
physical write total multi block          141,829         11.5           2.7
physical writes                           470,884         38.1           9.0
physical writes direct                      7,215          0.6           0.1
physical writes direct temporary            1,896          0.2           0.0
physical writes from cache                463,669         37.5           8.8
physical writes non checkpoint            448,198         36.3           8.5
rollback changes - undo records        48,219,732      3,902.0         916.6
```

```
rollbacks only - consistent read              6          0.0            0.0
summed dirty queue length                    92          0.0            0.0
transaction rollbacks                    48,268           3.            0.9
undo change vector size           3,295,395,408    266,669.5       62,644.2
user I/O wait time                       16,061          1.3            0.3
user calls                                5,596          0.5            0.1
user commits                              3,024          0.2            0.1
user rollbacks                           49,581          4.0            0.9
          -------------------------------------------------------------
```

Figure 15: Statistics Showing Undo Activity on the Disk Based System

Figure 1 In the above statistics we can see that we have 35.4 DBWR undo block writes per second out of 38.1 physical writes per second, so a majority of our write IO is to the undo segments. This is confirmed by examining the tablespace file IO in the tablespace report section of the AWR report. Writes to the undo segment are usually single block writes and there may be hundreds per second in a busy OLTP system or in a system where there are numerous small transactions. Generally, in a high transaction rate system the undo tablespace will be one of the top sources for writes in the database. This is shown for our example disk based system in Figure 16.

```
Tablespace
------------------------------
                  Av        Av       Av                 Av    Buffer    Av Buf
         Reads Reads/s    Rd(ms)  Blks/Rd    Writes Writes/s    Waits    Wt(ms)
-------------- ------- ------ ------- ------------ -------- ---------- ---
UNDOTBS1
           585       0     28.6      1.0    34,298        3          2      0.0
DATA
         9,044       1  3.8E+03     98.7    12,165        1          0      0.0
SYSAUX
        11,184       1     21.0      1.4     5,977        0          0      0.0
SYSTEM
         7,188       1     46.5      1.1     1,200        0         16     26.9
INDEXES
           599       0     28.3      1.0       581        0          0      0.0
USERS
           587       0     28.5      1.0       581        0          0      0.0
TEMP
           234       0      5.9      8.1       234        0          0      0.0
```

Figure 16: IO Based Statistics for the Disk Based System

In a ten minute sample we can see the IO timing for our system as it relates to the undo tablespace in Figure 17.

```
Date: 08/19/08                                                  Page:   1
Time: 10:13 AM         Snap 208 to 211 IO Timing Analysis           SYSTEM
                             aultdb1 database
FILENAME                                PHYRDS   PHYWRTS AVE_RD_TIM  AVE_WRT_TIM
------------------------------ ---------- ---------- ---------- ----------
+DATA2/datafile/sysaux.257.660755929  11184      5977      21.021       42.884
+DATA2/datafile/data.276.663265403      581       581      28.399       40.240
+DATA2/datafile/undotbs1.258.660755929  585     34298      28.598       37.923
+DATA2/datafile/system.256.660755927   7188      1200      46.503       25.725
+DATA2/datafile/users.259.660755929     587       581      28.466       22.495
+DATA/datafile/indexes.258.660765437    599       581      28.280       10.275
+DATA2/tempfile/temp.274.660989059       33        33       0.606        8.787
+DATA/datafile/data.259.660843403      3810     11003    4076.364        7.472
+DATA/datafile/data.257.660765277      4653       581    4042.241        7.263
+DATA2/tempfile/temp.273.660930207      200       200        6.8         6.95
```

Figure 17: IO Timings for a 10 Minute Sample of the Disk Based System

Without the "poisoning" of the undo writes, the IO profile for the transaction (a simple "INSERT INTO dup_lineitem SELECT * FROM h_lineitem) looks like the data in Figure 18.

```
Date: 08/19/08                                                  Page:   1
Time: 10:33 AM        Snap 122 to 123 IO Timing Analysis           SYSTEM
                              aultdb1 database

FILENAME                        PHYRDS   PHYWRTS   AVE_RD_TIM    AVE_WRT_TIM
------------------------------ -------- -------- ---------- -----------
+DATA2/sysaux.257.660755929         34      414         16.8          132.4
+DATA2/undotbs1.258.660755929       13      322         22.3           75.4
+DATA2/system.256.660755927         13       59         24.6           60.7
+DATA2/users.259.660755929          13       13         22.3           15.4
+DATA/data.259.660843403           468     5142         28.6           12.2
+DATA2/undotbs2.267.660756153       13       13         22.3           10.8
+DATA/indexes.258.660765437         13       13         20.8           10.0
+DATA/data.257.660765277           371       13         31.9            4.6
```

Figure 18: Simple Insert Stats (No-UNDO)

You can see that the read times on the DATA tablespace datafiles has dropped from over 4000 milliseconds to around 32 milliseconds. Obviously something must be done about the undo load. In the AWR excerpt in Figure 19 all datafiles and the undo tablespace have been moved to the Write Accelerator (the benefits of a small database!). Notice the drastic improvements in read times and that we have gone from 5 writes per second to 53 writes per second in the UNDO tablespace and reduced the buffer wait time for the undo tablespace from 345 milliseconds to 6 milliseconds.

```
Tablespace               Filename
------------------------ ----------------------------------------------
                   Av        Av        Av                 Av    Buffer    Av Buf
    Reads     Reads/s   Rd(ms)  Blks/Rd    Writes Writes/s     Waits    Wt(ms)
------------- ------- ------ ------- ------------ -------- ---------- ---
DATA                     /ssd1/oracle/tmsorcl/data01.dbf
   60,211           5      0.9    31.0      9,121        1          0       0.0
INDEXES                  /ssd1/oracle/tmsorcl/index01.dbf
    1,396           0      0.0     1.0      1,396        0          0       0.0
SYSAUX                   /ssd1/oracle/tmsorcl/tmsorcl/sysaux01.dbf
    4,670           0      0.1     1.4      4,038        0          0       0.0
SYSTEM                   /ssd1/oracle/tmsorcl/tmsorcl/system01.dbf
    2,128           0      0.1     1.0      2,004        0          0       0.0
TEMP                     /ssd1/oracle/tmsorcl/temp01.dbf
        0           0      N/A     N/A         81        0          0       N/A
UNDOTBS1                 /ssd1/oracle/tmsorcl/tmsorcl/undotbs01.dbf
    1,397           0      0.0     1.0    597,086       53        219       6.0
USERS                    /ssd1/oracle/tmsorcl/tmsorcl/users01.dbf
    1,396           0      0.0     1.0      1,396        0          0       0.0
          -------------------------------------------------------------
```

Figure 19: Same Undo Load, Only on Write Accelerator Based System

Notice that we have also completed over 9K writes to the data tablespace verses the 5K with the standard disk array, a 60% improvement in throughput for this test.

In addition there is the US enqueue that shows wait times for the undo segments due to serialization, and the KO-Multiple Object Checkpoint enqueue which shows the high checkpoint load from the undo operations. Figure 20 shows the enqueue report from a standard disk based system with significant undo load.

```
Enqueue Type (Request Reason)
------------------------------------------------------------------------------
     Requests  Succ Gets  Failed Gets   Waits  Wt Time (s)   Av Wt Time(ms)
------------ ------------ ----------- ----------- ------------ -----------
CF-Controlfile Transaction
       28,343     28,332          11      11            0            34.55
JS-Job Scheduler (queue lock)
      161,003    161,003           0       4            0            75.00
KO-Multiple Object Checkpoint (fast object checkpoint)
             9          9           0       1            0            30.00
PV-KSV slave startup (syncstart)
            65         65           0       1            0            30.00
           -------------------------------------------------------------
```

Figure 20: Disk Based System Enqueue Statistics with High Undo Load

Undo tablespace management has been automated to a fair amount; however, there are still a few tricks the savvy DBA can use to enhance the tuning of the undo segments by setting the transactions_per_rollback_segment and undo tablespace size. When undo segments are automatically managed, Oracle initially creates 10 segments then adds segments as the value of trunc (processes/transactions_per_rollback_segment) increases over 10. Notice there is no mention of transactions, just processes, so even if your database is a pure SELECT-only database you will still see undo segments brought online as this ratio increases.

Remember that a transaction changes data, thus a pure SELECT environment would have only internal transactions; no explicit or implicit external transactions such as INSERT, UPDATE, and DELETE environments would have. Generally this results in a few active undo segments with a bunch of offline inactive segments taking up space in the undo tablespace. By increasing or decreasing the value of transactions_per_rollback segment you can adjust the number of inactive segments generated. The initial size of the segments is dependent on the size of the tablespace. Figure 22 shows the undo segment section of the disk based system AWR report.

```
Undo Segment Summary                 DB/Inst: AULTDB/aultdb2  Snaps: 108-
111
-> Min/Max TR (mins) - Min and Max Tuned Retention (minutes)
-> STO - Snapshot Too Old count,  OOS - Out of Space count
-> Undo segment block stats:
-> uS - unexpired Stolen,  uR - unexpired Released,  uU - unexpired reUsed
-> eS - expired   Stolen,  eR - expired Released,    eU - expired   reUsed

Undo   Num Undo    Number of  Max Qry    Max Tx    Min/Max   STO       /uS/uR/uU/
 TS# Blocks (K) Transactions  Len (s) Concurcy  TR (mins)   OOS         eS/eR/eU
---- ---------- -------------- -------- -------- --------- ----- --------
   2      437.5       53,169   11,467        4   19.6/205   0/0   0/0/0/259/768/
             -------------------------------------------------------------
Undo Segment Stats                    DB/Inst: AULTDB/aultdb2  Snaps:
108-111
-> Most recent 35 Undostat rows, ordered by Time desc

             Num Undo    Number of  Max Qry  Max Tx  TunRet   STO      /uS/uR/uU/
  End Time      Blocks Transactions  Len (s)   Concy  (mins)   OOS        eS/eR/eU
  --------- ---------- ------------ -------- ------- ------- ----- ------------
  30-Aug 00:51   26,067        3,034   11,467       3     205   0/0    0/0/0/0/0/0
  30-Aug 00:41   38,889        4,491   10,866       3     195   0/0    0/0/0/0/0/0
  30-Aug 00:31   29,454        3,422   10,264       4     185   0/0    0/0/0/0/0/0
  30-Aug 00:21   11,134        1,517    9,663       4     175   0/0    0/0/0/0/0/0
  30-Aug 00:11    9,614        1,205    9,062       3     165   0/0    0/0/0/0/0/0
  30-Aug 00:01   11,919        1,663    8,461       4     155   0/0    0/0/0/0/0/0
  29-Aug 23:51   12,862        1,592    7,860       3     145   0/0    0/0/0/0/0/0
  29-Aug 23:41   14,413        1,747    7,258       3     135   0/0    0/0/0/0/0/0
```

```
29-Aug 23:31   12,631        1,558     6,657       4     125   0/0    0/0/0/0/0/0
29-Aug 23:21   12,641        1,543     6,055       3     115   0/0    0/0/0/0/0/0
29-Aug 23:11   11,946        1,731     5,453       3     105   0/0    0/0/0/0/0/0
29-Aug 23:01   12,229        1,640     4,852       4      95   0/0    0/0/0/0/0/0
29-Aug 22:51   11,852        1,509     4,251       3      85   0/0    0/0/0/0/0/0
29-Aug 22:41   10,631        1,328     3,650       3      75   0/0    0/0/0/0/0/0
29-Aug 22:31   38,927        4,485     3,049       4      65   0/0    0/0/0/0/0/0
29-Aug 22:21   39,064        4,486     2,448       3      55   0/0    0/0/0/0/0/0
29-Aug 22:11   40,183        4,748     1,847       3      45   0/0    0/0/0/0/0/0
29-Aug 22:01   37,295        4,357     1,246       4      35   0/0    0/0/0/0/0/0
29-Aug 21:51   37,819        4,848       935       3      30   0/0    0/0/0/0/0/0
29-Aug 21:41   17,888        2,265       334       3      20   0/0    0/0/0/0/0/0
          -------------------------------------------------------------
```

Figure 21: Undo Statistics From Standard Disk Based System

Why is it important to manage the growth of undo segments? Each time a segment is added, grown, or shrunk it results in wait events. Segments will be shrunk by the transaction that acquires them after they have grown beyond the calculated optimal size. Generally, unless your IO subsystem is already stressed, the extension of an undo segment is not overly expensive; however, if an undo segment is really large, the shrink operation can result in excessive wait times as the extents are cleaned up as this results in recursive SQL operations.

If there are no rollbacks, the writes to undo segments far exceed the reads from those segments. Thus, if the IO subsystem is already strained, the large number of undo writes in a heavy OLTP system will cause a degradation in overall performance. Moving the undo tablespace to its own RAID area can help with contention issues, but be sure that the RAID area used can handle the large number of concurrent writes that are required by the undo tablespaces.

So you have optimally sized and placed your undo segment tablespaces on their own RAID and yet, due to transaction load, you still see excessive wait times, then the Write Accelerator from Texas memory Systems will solve your problems. In tests using the Write Accelerator technology, moving the undo processing for the above test procedure to the Write Accelerator improved processing time by up to 50 percent or more, allowing 50 percent or more transactions to be processed in the same amount of clock time.

Section 5

Summary

In this paper we have examined the three major sources of write pain for the Oracle database: redo logs and undo and temporary tablespace operations. In the case of redo log activity you should first determine if the problem is truly IO related or is tied to CPU/internal scheduling conflicts. If the problem is IO subsystem related then the TMS Write Accelerator will help provide proper redo log acceleration. In the case of temporary tablespace IO issues, you need to first verify that the various parameters that control sort, hash, and bitmap memory usage are set correctly, and if the temporary tablespace still shows IO related issues, then the TMS Write Accelerator is indicated. Finally, undo writes, if they are a source of excessive waits after undo segments have been properly tuned, can be mitigated by placing undo tablespaces on the TMS Write Accelerator.

Many applications can see immediate improvements by moving all three sources of write waits, redo, temporary, and undo, to a high speed device such as solid state drives. After removing these sources of write contention, a careful analysis of the database can also reveal other sources of read/write issues such as hot indexes and tables that will also benefit from being moved to SSD. Proper application of write (and read) accelerator technology such as the TMS Write Accelerator will provide performance improvements of over 100%, as proved in many TPCH and TPCC based tests.

For more information on the Texas Memory Systems RamSan product line, please contact Texas Memory Systems at 713-266-3200 or visit www.superssd.com