An Oracle White Paper
November 2015

# Express Diagnosis of Oracle E-Business Suite Release 12 Upgrade Performance Issues

Executive Overview .......... 3
Introduction .......... 3
R12.2, Online Patching and Diagnostics .......... 4
Before Running the Oracle E-Business Suite Upgrade .......... 5
    Set Statistics Level = ALL .......... 5
    AWR Snapshot Interval and Retention Period .......... 6
During the Oracle E-Business Suite Upgrade .......... 6
    Obtain Top SQL in Cursor Cache or AWR .......... 6
    Obtain Display Cursor Report for Long-Running SQL .......... 7
    Obtain SQL Monitor Report for SQL Using Parallel Query/DML .......... 9
    Identify when SQL ran .......... 10
    Match Long-Running SQL to Jobs .......... 10
    Report on CBO Statistics for All Oracle E-Business Suite Tables 12
After the Oracle E-Business Suite Upgrade .......... 12
    Obtain AD Job Timing Report .......... 13
    Identify Long-Running Upgrade Jobs .......... 13
    Obtain File versions for Long-Running Jobs .......... 14
    Obtain AWR Reports .......... 15
    Run afxplain.sql to obtain statistics, parameters, and metadata... 15
    SQLT with XTRACT method .......... 16
Online Patching Diagnostics .......... 18
    Online Patching Enablement - Specific Diagnostics .......... 18
    Online Patching (ADOP) Logs and Diagnostics .......... 18
    Non ADOP Logs .......... 19
    Online Patching Log Analyzer Utility .......... 19
    adzdshowlog.out .......... 20
    Check the current status of the adop cycle .......... 20
Fixed Object and Dictionary Statistics .......... 20

## Executive Overview

This document describes the diagnostic strategies and methods that can be used during an Oracle E-Business Suite Release 12 upgrade to minimize downtime and expedite resolution of any issues.

The content applies to upgrades from 11.5.10, R12.0.n or R12.1.n to R12.2.n as well as upgrades from 11.5.10 to R12.1.3.

The same principles can also be applied to other Oracle E-Business Suite upgrades.

For more detailed information see the My Oracle Support document “Best Practices for Minimizing Oracle E-Business Suite Release 12 Upgrade Downtime (Document 1581549.1)”

## Introduction

When analyzing Release 12 upgrade performance Issues, the goal is to:

- Prevent wasted test iterations and effort, by aiming to provide solutions that resolve issues on the first attempt.
- Maximize the number of performance issues resolved in the time available.

An important point is that upgrade jobs cannot be tested in isolation: they will be tested on the next test iteration. If a fix for a performance issue does not work, then that is a potential wasted test iteration.

To do this the following are required:

- Actual statistics: So it is possible to see exactly which execution plan steps are inefficient, rather than those that might be inefficient. The likely impact of a performance fix can also be estimated. There will then be a higher probability of providing a fix that can solve the performance issue first time. It will also be possible to identify marginal fixes (i.e. fixes that reduce elapsed times by 10-50%, for example by having more specific index access). These fixes often reduce contention between workers.

- Diagnostics that are quick and easy to run, so that diagnostics can be obtained faster and on more jobs/SQL.

- Diagnostics that have very little impact on the performance of the Release 12 upgrade: if they can be run during the upgrade then the results are obtained sooner and the issue resolved more quickly.

## R12.2, Online Patching and Diagnostics

The new diagnostics available for online patching (ADOP) are logs. They do not identify the SQLs or underlying events that caused the performance issue.

Online Patching (ADOP) and the underlying Edition-Based Redefinition are only used after “Online Patching Enablement” for the “R12.AD.C.Delta.n”, “R12.TXK.C.Delta.n”, “12.2.n RUP” and subsequent patches.

They may introduce new performance challenges, particularly with internal SQL and ADOP SQL (i.e. SQL run from or on AD_ZD objects).

However, in order to diagnose any performance issues, it is still important to obtain Display Cursor Reports, SQL Monitor Reports, AWR Reports, AD Job Timing Reports, SQLT XTRACTs etc.

## Before Running the Oracle E-Business Suite Upgrade

### Set Statistics Level = ALL

alter system set statistics_level='ALL'

This is the simplest way to see actual row source statistics (including elapsed time, physical reads, buffer gets etc) for each execution plan line (on SQLT and Display Cursor report). The alternative of SQL Trace and TKPROF requires editing standard code.

Using this strategy will typically speed up the resolution of issues significantly and may also allow the correct solution to be identified first time.

Alternatively, the same actual execution statistics can be collected by setting the initialization parameter _rowsource_execution_statistics=TRUE (with statistics_level = 'TYPICAL'). This gives a lower overhead than statistics_level=ALL.

Some technical architects and DBAs at customers (or implementing partners) can be resistant to setting statistics_level = ALL (or _rowsource_execution_statistics = TRUE), believing that this can slow down performance significantly.

Two points are relevant here:

- Although setting STATISTICS_LEVEL = ALL / _rowsource_execution_statistics = TRUE will have some performance impact, it is likely to be small and not significant. The Release 12 upgrade is made up of batch processes, and so the statistics workload is a much lower proportion of the total.
- Even if the performance impact is significant, the goal is to reduce the elapsed times for the latter dry runs and go live (when it will be feasible to revert STATISTICS_LEVEL / _rowsource_execution_statistics to their previous values). So seeing an increase in elapsed time during an early stage of testing is not an issue.

So there may be a small impact on elapsed time and the work that has to be done initially, but it will help to subsequently reduce the elapsed time and amount of re-work that needs to be done.

Note that setting statistics_level = ALL while Automatic Workload Repository (AWR) is enabled could significantly increase the number of rows inserted to the WRH$_LATCH_CHILDREN table. So monitor the SYSAUX tablespace to ensure that it does not run out of space.

### AWR Snapshot Interval and Retention Period

Automatic Workload Repository (AWR) should be enabled with a snapshot of 30 minutes (the default is 60 minutes). For short upgrades, a shorter snapshot may be more suitable.

The AWR retention period should be long enough to cover the duration of the upgrade run and a significant period afterwards (to gather diagnostics and analyze). The suggestion is N+7 days, where N is the estimated upgrade time, but a longer period will provide more time to gather subsequent diagnostics and statistics.

## During the Oracle E-Business Suite Upgrade

### Obtain Top SQL in Cursor Cache or AWR

This could be internal or application SQL.

These scripts should be run regularly during the upgrade, particularly when there are long-running jobs.

If SQL is still in memory (cursor cache) the following can be used to identify long running SQLs that may not have been written to the AWR yet (at last snapshot):

```
SELECT * FROM
(SELECT
   ss.sql_id,
   ROUND(SUM(ss.elapsed_time/1000000),0)  elapsed_time_secs,
   ROUND(SUM(ss.cpu_time/1000000),0)  cpu_time_secs,
   SUM(ss.disk_reads)  disk_reads,
   SUM(ss.direct_writes)  direct_writes,
   SUM(ss.buffer_gets)  buffer_gets,
   SUM(ss.px_servers_executions)  px_server_execs,
   SUM(ss.rows_processed)  rows_processed,
   SUM(ss.executions)  executions,
   SUM(ss.application_wait_time)  apwait_secs,
   SUM(ss.sharable_mem)  sharable_mem,
   SUM(ss.total_sharable_mem)  total_sharable_mem
 FROM v$sqlstats  ss
 GROUP BY  ss.sql_id
 ORDER BY 2  DESC)
 WHERE ROWNUM  <= 100;
```

The following SQL script will report the longest running SQL between two AWR snapshots:

```
SELECT * FROM
(SELECT
dhs.sql_id,
ROUND(SUM(dhs.elapsed_time_delta/1000000),0) elapsed_time_secs,
ROUND(SUM(dhs.cpu_time_delta/1000000),0) cpu_time_secs,
SUM(dhs.disk_reads_delta) disk_reads,
SUM(dhs.buffer_gets_delta) buffer_gets,
SUM(dhs.px_servers_execs_delta) px_server_execs,
SUM(dhs.rows_processed_delta) rows_processed,
SUM(dhs.executions_delta) executions,
ROUND(SUM(dhs.iowait_delta/1000000),0) iowait_secs,
ROUND(SUM(dhs.clwait_delta/1000000),0) clwait_secs,
ROUND(SUM(dhs.ccwait_delta/1000000),0) ccwait_secs,
ROUND(SUM(dhs.apwait_delta/1000000),0) apwait_secs
FROM dba_hist_sqlstat dhs
,v$database d
WHERE dhs.dbid = d.dbid
AND snap_id > <begin snap> and snap_id <= <end snap>
GROUP BY dhs.sql_id
ORDER BY 2 DESC)
WHERE ROWNUM <= 100;
```

Where <begin snap> and <end snap> are the start and end snapshot IDs.

The output of this statement will look similar to the following:

```
SQL_ID        ELAPSED_TIME_SECS CPU_TIME_SECS   DISK_READS  BUFFER_GETS ….
------------- ----------------- --------------- ----------  ----------- ….
5vaxut40xbrmr            367440           42999   34838244   3795838289 ….
943ra4b7zg28x            264369          170788     441127    562033013 ….
fkkrk9frwqfdr             70370            6448    3599284    469639133 ….
4847s6dt6sds9             68298           38896    7125573   1327384554 ….
2k3uw8n473r30             63600           27402   20043712    587615960 ….
```

The elapsed time is the maximum elapsed time for all workers of a job.

Enterprise Manager can also be used to identify expensive SQL as it occurs.

## Obtain Display Cursor Report for Long-Running SQL

For long-running SQL reported by the above script, run a display cursor report (with ALL +ALLSTATS option).

This displays the execution plan of any cursor loaded in the cursor cache. At a basic level, it shows the runtime execution plan. However, the format ALL also includes extra information such as pruning, parallel execution, predicate, projection, alias and remote SQL information.

The +ALLSTATS option (which includes IOSTATS and MEMSTATS) will include actual statistics for each execution plan step. These include:

- Elapsed time
- Physical reads
- Buffer gets
- Memory used (in PGA) for memory intensive operations (such as hash joins, sorts, and bitmap operators).

However, this additional information is only provided if STATISTICS_LEVEL=ALL / _rowsource_execution_statistics = TRUE.

*Note that SQLT with XTRACT will also report actual row source statistics in the same circumstances. However, display_cursor provides a simpler view of the information. It can also be run during the upgrade, while the long running SQL is in progress, without much of an overhead.*

*Note that the display cursor report should be run as soon as possible. If it is delayed, the cursor may have been flushed from memory or invalidated. In such a case, no data will be available.*

The report can be produced by running the following SQL script:

```
SET pages 0
SET lines 300
SET LONG 10000
SET LONGCHUNKSIZE 10000
SPOOL<report_name>.txt
SELECT * FROM TABLE(dbms_xplan.display_cursor('<sql_id>', NULL, 'ALL +ALLSTATS'));
SPOOL OFF;
```

For more information see the "Display Cursor" section in My Oracle Support document "Oracle E-Business Suite Performance Guide (Document 1672174.1)".

If the SQL is no longer in memory, but is in the AWR, use the Display AWR report instead:

```
SET pages 0
SET lines 300
SET LONG 10000
SET LONGCHUNKSIZE 10000
SPOOL<report_name>.txt
SELECT * FROM TABLE(dbms_xplan.display_awr('<sql_id>', NULL, NULL, 'ALL'));
SPOOL OFF;
```

Be aware that the Display AWR report (DBMS_XPLAN.DISPLAY_AWR) does not report on actuals: it does not have a +ALLSTATS option, and there are no actual statistics for execution plan steps stored in AWR.

Also note that the display cursor and AWR reports only show the sql_text (first 1000 characters) and not the sql_fulltext. So, if necessary, run the following SQL script to obtain the full SQL text:

```
SET pages 0
SET lines 300
SET LONG 10000
SET LONGCHUNKSIZE 10000
SPOOL<report_name>.txt
SELECT sql_id, sql_text, sql_fulltext FROM v$SQL
WHERE sql_id = '<sql_id>';
SPOOL OFF;
```

### Obtain SQL Monitor Report for SQL Using Parallel Query/DML

Obtain SQL Monitor Reports for SQL that uses Parallel Query or DML

The main advantage of this is that it gives a good view of how parallel SQL/DML performs across stages of the plan and parallel slaves.

It can also give a good idea of the actual executions and row counts for each execution plan line even if "statistics_level" initialization parameter is not set to ALL ( or "_rowsource_execution_statistics" is not set to TRUE) at the time the SQL is executed.

It can be run during the upgrade, while the long running SQL is in progress, without much of an overhead.

Produce this report by running the following SQL script:

```
set trimspool on
set trim on
set pages 0
set long 10000000
set longchunksize 10000000
set linesize 200
set termout off
spool sql_monitor_for_<sql_id>.htm
variable my_rept CLOB;
BEGIN
:my_rept := dbms_sqltune.report_sql_monitor(sql_id => '<sql_id>', report_level =>
'ALL', type => 'HTML');
END;
/
print :my_rept

spool off;

set termout on
```

For more information see the "SQL Monitor Report" section in My Oracle Support document "Oracle E-Business Suite Performance Guide (Document 1672174.1)"

## Identify when SQL ran

The following SQL will show when a particular piece of SQL ran (i.e. between which snapshots). This is useful in matching SQLs to jobs:

```
SELECT
dhs.sql_id,
dsn.snap_id,
dsn.begin_interval_time,
dsn.end_interval_time,
ROUND(SUM(dhs.elapsed_time_delta/1000000),0) elapsed_time_secs
FROM dba_hist_sqlstat dhs
,v$database d
,dba_hist_snapshot dsn
WHERE dhs.dbid = d.dbid
AND dsn.snap_id = dhs.snap_id
AND dsn.dbid = dhs.dbid
AND dsn.instance_number = dhs.instance_number
AND dhs.sql_id = '<sql_id>'
AND dsn.snap_id > <begin_snap> and dsn.snap_id <= <end_snap>
GROUP BY dhs.sql_id, dsn.snap_id, dsn.begin_interval_time, dsn.end_interval_time
ORDER BY dsn.snap_id;
```

Where <begin snap> and <end snap> are the start and end snapshot IDs.

The output of this statement will look similar to the following:

```
SQL_ID        SNAP_ID BEGIN_INTERVAL_TIME     END_INTERVAL_TIME       ELAPSED_TIME_SECS
------------- ------- ----------------------- ----------------------- -----------------
2k3uw8n473r30 8278    04-JAN-13 23.00.25.5560 05-JAN-13 00.00.21.1620             23123
2k3uw8n473r30 8279    05-JAN-13 00.00.21.1620 05-JAN-13 01.00.38.2680             37145
```

## Match Long-Running SQL to Jobs

The following SQL will list jobs running at any point between two time intervals, with the longest running jobs being shown first. This is useful in matching SQL to jobs.

Where:

<start_time> = start of period to report in format YYYYMMDDHH24MISS

<end_time> = end of period to report in format YYYYMMDDHH24MISS

Note that the job must have completed before it will be reported by this SQL script.

```
SELECT
phase,
phase_name,
product,
job_name,
max_elapsed_time,
min_start_time,
max_end_time,
workers
FROM
(SELECT
   phase,
   phase_name,
   product,
   job_name,
   MAX(elapsed_time) elapsed_time_unconv,
   LPAD(FLOOR(MAX(elapsed_time)*24), 4)||':'||
   LPAD(FLOOR((MAX(elapsed_time)*24-floor(MAX(elapsed_time)*24))*60), 2, '0')||':'||
   LPAD(MOD(ROUND(MAX(elapsed_time)*86400), 60), 2, '0')
   max_elapsed_time,
   INITCAP(TO_CHAR(MIN(start_time),' MON DD HH24:MI',
               'NLS_DATE_LANGUAGE = american')) min_start_time,
   INITCAP(TO_CHAR(MAX(end_time),' MON DD HH24:MI',
               'NLS_DATE_LANGUAGE = american')) max_end_time,
   count(worker_id) workers
FROM ad_task_timing
WHERE session_id = <session_id>
AND
(
start_time BETWEEN TO_DATE('<start_time>','YYYYMMDDHH24MISS')
           AND TO_DATE('<end_time>','YYYYMMDDHH24MISS')
OR
NVL(end_time, start_time+elapsed_time)
            BETWEEN TO_DATE('<start_time>','YYYYMMDDHH24MISS')
            AND TO_DATE('<end_time>','YYYYMMDDHH24MISS')
)
GROUP BY phase, phase_name, product, job_name)
ORDER BY elapsed_time_unconv DESC;
```

The output of this statement will look similar to the following:

```
      Phase                            max elapsed
phase name   product job_name          time        min_start_time max_end_time workers
----- ------ ------- ----------------- ----------- -------------- ------------ -------
255   upg+80 zx      zxaptrxmigupd.sql     2:43:47 Mar 13 04:22   Mar 13 07:06      64
255   upg+80 ap      apxlaupg.sql          1:38:57 Mar 13 04:03   Mar 13 05:42       1
```

To find upgrade AD jobs that are in progress use adctrl option 1 (Show worker status).

When jobs started can be determined by looking at the patch log file. For example:

```
$ cat u_merged.log|grep -A2 cstpostimportaad.sql
```

```
Assigned: file cstpostimportaad.sql on worker 48 for product bom username BOM.
Time is: Fri Mar 22 2013 22:48:54
```

### Report on CBO Statistics for All Oracle E-Business Suite Tables

Report on the CBO statistics for Oracle E-Business Suite tables during the upgrade, before adsstats.sql is run (in R12.1.1 or R12.2.0 driver).

The script adsstats.sql will populate the statistics correctly before the end of the upgrade. The fact that tables may have incorrect statistics during the upgrade will not be visible. So it may not be possible to see that a table had null, zero or inaccurate CBO statistics, and that this is the reason for an expensive execution plan.

The following script will report on the CBO statistics for all Oracle E-Business Suite tables:

```
SELECT owner, table_name, num_rows,
TO_CHAR(last_analyzed,'DD-MON-YYYY HH24:MI:SS') last_analyzed
FROM all_tables
WHERE owner IN
(SELECT upper(oracle_username) sname
FROM   fnd_oracle_userid
WHERE  oracle_id BETWEEN 900 AND 999
AND read_only_flag = 'U'
UNION ALL
SELECT DISTINCT upper(oracle_username) sname
FROM            fnd_oracle_userid a,
fnd_product_installations b
WHERE           a.oracle_id = b.oracle_id
)
ORDER BY owner, table_name;
```

The output of this statement will look similar to the following:

```
OWNER                    TABLE_NAME                    NUM_ROWS LAST_ANALYZED
---------------------- ---------------------------- ---------- ------------------------
ABM                    ABM_ACC_MAP_SUM_REP                   0 06-JAN-2013 08:46:33
ABM                    ABM_ACT_ACC_RU_DAT                    0 06-JAN-2013 08:46:35
ABM                    ABM_ACT_STA_RU_DAT                    0 06-JAN-2013 08:46:36
ABM                    ABM_ACT_TAGS                          0 06-JAN-2013 08:46:37
ABM                    ABM_API_TEMPLATES                    38 06-JAN-2013 08:44:53
ABM                    ABM_API_TEMPLATES_TL                722 06-JAN-2013 08:41:16
ABM                    ABM_API_TEMPLATE_ATTRIBUTES         248 06-JAN-2013 08:44:34
```

## After the Oracle E-Business Suite Upgrade

These are diagnostics to be obtained directly after each of the main patches/RUPs have completed.

This will be directly after 12.1.1 and 12.1.3 upgrades for an upgrade to Release 12.1.3 and directly after each of 12.2.0 Upgrade, Online Patching Enablement, R12.AD.C.Delta.n, R12.TXK.C.Delta.n and 12.2.n RUPs for an upgrade to Release 12.2.n.

## Obtain AD Job Timing Report

This reports the top 100 time consuming jobs. Along with failed, deferred, re-started, skipped jobs, and the timing of upgrade phases.

When ADOP, AutoPatch or AD Administration is run, an AD Job Timing report (adt<session_id>.lst) is automatically generated.

The contents of this report can be accessed from Oracle Application Manager, or reports can be obtained for completed upgrade sessions from the APPL_TOP/admin/<SID>/out directory. The report is called adt<session_id>.lst.

The AD Job Timing Report can also be run for AD Administration jobs from the command line.

```
$ cd $APPL_TOP/admin/<SID>/out

$ sqlplus <APPS username>/<APPS password> @$AD_TOP/admin/sql/adtimrpt.sql \

<session id> <output file>
```

Where <session_id> is the session of the timing statistics required, and <output file> is the name of the file where the statistics will be written.

$AD_TOP/admin/sql/adtimdet.sql can also be run in a similar way. This gives details on all jobs ordered by phase or elapsed time. This is useful for finding out how long any job took to run, and also where the "Top 100 Time Consuming Jobs" is dominated by multiple workers of a few jobs.

Note that the SQL scripts may be in $AD_TOP/sql (not admin/sql).

See "Oracle E-Business Suite Maintenance Guide" for more information.

## Identify Long-Running Upgrade Jobs

Note that the "Top 100 Time Consuming Jobs" section of the standard adtimrpt.sql report, lists all workers for AD Parallel jobs separately. So the top 100 can be dominated by a handful of jobs.

The following SQL can be used to list all jobs in order of maximum elapsed time (descending), but reporting all workers of an AD Parallel job in one line. It only reports completed jobs.

Where <session_id> is the ID for the upgrade session and not a user session.

Be aware of jobs that are called multiple times in the same phase (e.g. akload.class, LoadMap.class, XDOLoader.class).

```
SELECT
phase,
phase_name,
product,
job_name,
max_elapsed_time,
min_start_time,
max_end_time,
workers
FROM
(SELECT
   phase,
   phase_name,
   product,
   job_name,
   MAX(elapsed_time) elapsed_time_unconv,
   LPAD(FLOOR(MAX(elapsed_time)*24), 4)||':'||
   LPAD(FLOOR((MAX(elapsed_time)*24-floor(MAX(elapsed_time)*24))*60), 2, '0')||':'||
   LPAD(MOD(ROUND(MAX(elapsed_time)*86400), 60), 2, '0')
   max_elapsed_time,
   INITCAP(TO_CHAR(MIN(start_time),' MON DD HH24:MI',
               'NLS_DATE_LANGUAGE = american')) min_start_time,
   INITCAP(TO_CHAR(MAX(end_time),' MON DD HH24:MI',
               'NLS_DATE_LANGUAGE = american')) max_end_time,
   count(worker_id) workers
FROM ad_task_timing
WHERE session_id = <session_id>
GROUP BY phase, phase_name, product, job_name)
ORDER BY elapsed_time_unconv DESC;
```

The output of this statement will look similar to the following:

```
  Phs                                      MaxElpsd   Min           Max            Num
  Num Phase        Prod  Job               Time       Start Time    End Time       Wrkr
----- ------------ ----- ----------------- ---------- ------------- ------------- -----
  255 upg+80       zx    zxaptrxmigupd.sql    2:43:47 Mar 13 04:22  Mar 13 07:06     64
  351 last+63      ad    adsstats.sql         2:09:42 Mar 13 11:14  Mar 13 13:23      1
  255 upg+80       ap    apxlaupg.sql         1:38:57 Mar 13 04:03  Mar 13 05:42      1
  251 upg+76       ap    apilnupg.sql         0:55:45 Mar 13 01:24  Mar 13 02:19     64
  241 upg+57       ont   ontup251.sql         0:25:07 Mar 13 00:11  Mar 13 00:36      1
….
```

## Obtain File versions for Long-Running Jobs

When an SR or Bug is raised, Oracle Support and Development will ask for the version of the job (file) that has the issue, or the version of code (file) used by the job.

Be sure to identify the version of the files used in the pack/patch where the issue occurs.

For example, for an R12.2.n upgrade, the version present in the file system or database after all the upgrade steps – including R12.2.n RUP and post upgrade steps – may be different from the one used during the R12.2.0 upgrade. Similarly, the version present after a R12.1.3 upgrade may be different from the version used in the R12.1.1 driver.

The best place to search is in the driver (or merged driver) file for the relevant pack/patch (e.g. $ cat u_merged.drv|grep –A5 cstpostimportaad.sql).

## Obtain AWR Reports

See My Oracle Support document "Performance Diagnosis with Automatic Workload Repository (Document 1674086.1)" for more information.

Get AWR reports for:

- The whole period that the upgrade is running.
- For the duration of long-running jobs (i.e. between the snapshots taken just before the job starts and just after it finishes).
- Each individual snapshot.

awrrpt.sql will typically be used to generate the AWR reports. Always choose HTML report type. On an Oracle RAC instance, awrrpti.sql will usually suffice, as the upgrade will be run on one Oracle RAC node only.

AWR reports can be automated. This is useful if producing a large number of AWR reports, particularly for successive snapshots. See the "Automating AWR Reports" section in My Oracle Support document "Performance Diagnosis with Automatic Workload Repository (Document 1674086.1)".

Note that some fixed objects and dictionary objects (particularly WRH$_LATCH_CHILDREN, especially if statistics_level = ALL, or there is a high retention period or a short snapshot interval) will have grown significantly during the upgrade. So, fixed object and dictionary statistics may need to be gathered before running AWRs.

## Run afxplain.sql to obtain statistics, parameters, and metadata

$FND_TOP/sql/afxplain.sql can be used to produce some of the same output as SQLT.

It does not require any installation on the environment (other than the script file itself) and has a negligible performance impact. It can be run whilst the upgrade is in progress.

It provides the following information:-

- Database/Apps version, O/S version and instance/database name
- Explain Plan with cost and predicate information
- Amount of shared memory
- Object information including Package, View and Triggers
- Statistics for all objects referenced by SQL (including columns and indexes)
- Non-default Database parameters (init.ora)

Note that the output will not contain the following:

- Actual counts of rows on each table.
- Actual execution plan, but this should be available from the display_cursor (or display_awr).

Create a file containing the SQL that the diagnostics are required for. In this example it is called query.sql.

Run:

```
afxplain.sql query.sql Y N
```

The Y N parameters respectively specify that a CBO Trace (10053) is required, but that statistics will not be exported.

The output will be in file *query.sql.out*

The trace for the 10053 event will be in the user dump destination.

For more information see the "SQLT" > "Alternative to SQLT (afxplain.sql)" section in My Oracle Support document "Oracle E-Business Suite Performance Guide (Document 1672174.1)"

### SQLT with XTRACT method

The SQLT with XTRACT method can provide the same information (as afxplain.sql) and more, particularly the actual table counts and the actual execution plan. So   this may be run for some of the longest-running SQL (the sql_ids of these will have been obtained in the steps above).

However, it will take much longer to run.

If the SQL was executed when "statistics_level" was set to ALL (or "_rowsource_execution_statistics" is set to TRUE) then it will contain actual row source statistics (I/O, buffer gets, elapsed time) on the execution plan (provided the cursor is still in memory (cursor cache)).

The SQLT should be provided on the same environment where the performance issue was observed, and should be run as soon after the relevant program/process as possible.

Be aware of any actions that may alter the data that SQLT is capturing (that is, actions that take place after the observed performance issue, but before SQLT is run). For example, statistics being gathered, removed, locked, imported, or data being deleted from temporary or interface tables.

See the following My Oracle Support documents for more information:

- Oracle E-Business Suite Performance Guide (Document 1672174.1), section "SQLT"
- All About the SQLT Diagnostic Tool (Document 215187.1)

Note that SQLT runs AWR and ASH reports. Some dictionary objects (particularly WRH$_LATCH_CHILDREN, especially if statistics_level is set to ALL) will have grown significantly during the upgrade. So, it may be necessary to gather fixed object and dictionary statistics before running SQLT.

SQLT can take quite a while to run. To reduce the workload, it is recommended that the following are run (from SQL*Plus) before running sqltxtract.sql:

To disable Test Case Builder (TCB) and SQL Tuning Advisor:

```
EXEC sqltxplain.sqlt$a.set_param('test_case_builder', 'N');

EXEC sqltxplain.sqlt$a.set_param('sta_time_limit_secs', '30');
```

To disable the automatic export of a test case repository:

```
EXEC sqltxplain.sqlt$a.set_param('export_repository', 'N');
```

If SQLT still takes a long time, and the schema objects used by the SQL contain a large number of subpartitions, the granularity of the data collected for segment, column and histogram statistics can be reduced as follows:

```
EXEC sqltxplain.sqlt$a.set_param('c_gran_segm', 'PARTITION');

EXEC sqltxplain.sqlt$a.set_param('c_gran_cols', 'PARTITION');

EXEC sqltxplain.sqlt$a.set_param('c_gran_hgrm', 'PARTITION');
```

Note that these commands can all be run as APPS. They do not need to be run as user SQLTXPLAIN.

These values are stored in a table, SQLTXPLAIN.SQLI$_PARAMETER. Once they are set, they do not need to be re-set for each execution of SQLT. The current values can be checked by querying this table.

To reduce the time further the counting of rows on tables can be disabled, by running the following. However, information on the actual number of rows in each table will be lost.

```
EXEC sqltxplain.sqlt$a.set_param('count_star_threshold', '0');
```

All of this assumes that a SQLT version greater than 1.4.4.4 (April 2, 2012) is being used.

## Online Patching Diagnostics

### Online Patching Enablement - Specific Diagnostics

The Online Patching Enablement patch is applied using AutoPatch (adpatch). In addition to the general diagnostics above the output from the following script will be useful during Online Patching Enablement:

```
$ sqlplus apps @$AD_TOP/sql/ADZDSHOWDDLS.sql
```

### Online Patching (ADOP) Logs and Diagnostics

All the ADOP logs are located on the non-editioned file system (fs_ne) in the <INSTALL BASE>/fs_ne/EBSapps/log/adop directory e.g.

```
/u01/PROD/fs_ne/EBSapps/log/adop
```

Each cycle of ADOP creates a subdirectory corresponding to the patch session ID, e.g.

```
/u01/PROD/fs_ne/EBSapps/log/adop/n
```

Where n is the session ID.

It is easiest and quickest to produce a zip of the entire directory.

The main files of interest are the ADOP logs (e.g. adop_YYYYMMDD_HHMISS.log).

But the adzdshowlog.out, adworker*.log, u*.log, u*.lgi, admrgpch*.log files are all useful and under the same path.

When running ADOP the on screen terminal output will mention which ADOP session ID is in use.

```
e.g. /u01/PROD/fs_ne/EBSapps/log/adop/9/apply_20121011_024437
```

The session ID directory will contain a trace file for each phase (e.g. adop_20130316_091340.log) and a corresponding log directory for each phase containing other logs (e.g. apply_20130316_091340).

The timestamp of the trace file and the corresponding log directory will match.

## Non ADOP Logs

The same log directory for each phase (e.g. apply_20130316_091340) also contains some AD Utility and worker logs.

These include adrelink.log, adlibin.log, adlibout.log, adworknnn.log. The most useful are the adworknnn.log files that show the jobs run on each AD Parallel worker along with timestamps.

## Online Patching Log Analyzer Utility

This is delivered in R12.AD.C.Delta.n (since R12.AD.C.Delta.4).

This utility analyzes adop log directories for errors and warnings, and displays messages to help the user quickly identify any problems that may have occurred during an adop run. It thereby offers an alternative to reviewing log files manually.

The Log Analyzer utility can be run without options:

To scan all log directories of the latest adop session for errors:

```
$ adopscanlog
```

The utility can also be run with various options. Examples include:

To scan log directories relating to the latest run of adop in the latest session:

```
$ adopscanlog -latest=yes
```

To scan log directories relating to the latest run of the specified phase, in the latest session:

```
$ adopscanlog -latest=yes -phase=<phase_name>
```

To scan all log directories of a given session (represented by a session_id) for errors:

```
$ adopscanlog -session_id=<number>
```

To see a complete list of supported parameters:

```
$ adopscanlog -help
```

### adzdshowlog.out

This reports the contents of the AD_ZD_LOGS table. This contains messages on the progress of online patching with timestamps. The contents of this table will be truncated every time cleanup/prepare phase is run.

This can also be obtained for previous phases by running the following script:

```
$ sqlplus apps @$AD_TOP/sql/ADZDSHOWLOG.sql
```

Or running the SQL

```
SELECT * FROM ad_zd_logs ORDER BY log_sequence desc;
```

### Check the current status of the adop cycle

Source the run filesystem environment file and run command

```
adop -status
```

Usage:

adop -status generates a summary report

adop -status <sessionID> generates a summary report for that session ID

adop -status -detail generates a detailed report

## Fixed Object and Dictionary Statistics

There can sometimes be issues with Fixed Object and Dictionary Statistics in Online Patching Enablement, R12.2.n RUPs (online patching) or when producing AWR reports or SQLT.

A fixed object (X$ table) resides in memory only, and typically records the information about the instance or memory structures. The v$ dynamic performance views are defined on top of X$ tables e.g. V$SQL and V$SQL_PLAN.

Data dictionary tables (e.g. SYS.USER$, SYS.TS$, SYS.SEG$, SYS.OBJ$, SYS.TAB$, SYS.FILE) are stored on data files like normal application tables.

These objects may have grown due to an upgrade/patch (e.g. R12.2.0 upgrade).

There are also changes to fixed and dictionary objects due to Online Patching Enablement (and the underlying Edition-Based Redefinition). As a result internal SQL in Online Patching Enablement, R12.2.n RUPs and other online patches can sometimes be long running. Gathering fixed object or dictionary statistics can help in these circumstances. Particularly on editioning objects.

There may be additional specific circumstances during the upgrade where fixed object or dictionary statistics need to be gathered (such as before importing schema statistics or running SQLT or AWR reports when AWR has grown significantly).

If there is internal SQL (on V$ views or on SYS/SYSTEM objects) appearing high in AWR and TKPROF reports, it is likely that dictionary and fixed object statistics need to be gathered.

Note that the FND_STATS API does not gather statistics for dictionary or fixed objects. The DBMS_STATS APIs need to be used.

The commands are:

```
execute DBMS_STATS.GATHER_FIXED_OBJECTS_STATS(no_invalidate=>FALSE);

execute DBMS_STATS.GATHER_DICTIONARY_STATS( estimate_percent =>
DBMS_STATS.AUTO_SAMPLE_SIZE, options => 'GATHER AUTO',
no_invalidate=>FALSE);
```

*Usually the "no_invalidate=>FALSE" argument will not be needed. However, the procedures DBMS_STATS.set_database_prefs, set_global_pefs, set_schema_prefs or set_table_prefs could have been used to set the default value for NO_INVALIDATE to TRUE.*

If there are only a handful of internal SQLs with inefficient execution plans and only a few objects then specific objects could be targeted rather than gathering all dictionary or fixed object statistics.

e.g.

```
exec dbms_stats.gather_table_stats(ownname=>'SYS', tabname=>'OBJ$',
no_invalidate=>false);

exec dbms_stats.gather_table_stats(ownname=>'SYS', tabname=>'X$KQFP',
no_invalidate=>false);
```

ORACLE

Oracle is committed to developing practices and products that help protect the environment

Express Diagnosis of Oracle E-Business Suite
Release 12 Upgrade Performance Issues
November 2015
Author: Jim Machin

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200

oracle.com

Hardware and Software, Engineered to Work Together