Oracle
Performance Scalability Reliability

Oracle Fusion Middleware 11g SOA White Paper
February 2013

SOA 11g Database Performance

ORACLE

Disclaimer

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Change Record

Date	Author	Version	Change Reference
03-Jan-2012	Srinivas Kasam	1.0	Initial Draft
25-July-2012	Srinivas Kasam	1.0	Updated the doc with new topics
18-Dec-12	Srinivas Kasam	1.0	Incorporated Dillip's feedback.
11-Jan-13	Srinivas Kasam	1.0	Incorporated Lester's feedback.
15-Feb-13	Srinivas Kasam	1.0	Incorporated feedback from other reviewers.

Contributors

Contributor	Role
Srinivas Kasam	Principal Member Technical Staff, PSR Engineering

Reviewers

Name	Role	Document Status	Date Reviewed	Comments Incorporated
Dillip Praharaj	Architect, Database Performance, PSR Engineering	Review	11/12/12	Yes
Lester	Fusion DBMS Architect, PSR Engineering	Review	01/04/13	Yes
Nitin Jain	Sr. Director, PSR Engineering	Review	01/16/2013	Yes
Michael Bousamra	Principal Member of Technical Staff	Review	01/29/2013	Yes

Table of Contents

Overview:

This Fusion Middleware 11g white paper discusses performance problems that could arise commonly in a SOA database and also gives recommendations on how to resolve those performance problems. These recommendations are for SOA 11g running under Oracle database release 11.1.0.1 and above. SOA db can be hosted in a non- RAC configuration and RAC configuration.

Tuning Database Parameters:

For applications that access SOA database, ensure that your database is properly configured to support your application's requirements.

The following table provides the database initialization parameter guidelines for SOA databases. These parameter values are intended to provide a baseline. As your deployment and workloads characteristics change, these values may need to be adjusted as well.

Parameter Name	Description, Default value	Recommended start value
audit_trail	Enables or disables database auditing. Default value: DB.	If there is NO policy to audit db activity, consider setting this parameter to NONE. Enabling auditing can impact performance
plsql_code_type	PLSQL_CODE_TYPE specifies the compilation mode for PL/SQL library units. • INTERPRETED: PL/SQL library units are compiled to PL/SQL bytecode format. Such modules are executed by the PL/SQL interpreter engine. • NATIVE: PL/SQL library units are compiled to native (machine) code. Such modules are executed natively without incurring any interpreter impacts. Default value: INTERPRETED.	NATIVE
nls_sort	NLS_SORT specifies the collating sequence for ORDER BY queries. • If the value is set to BINARY, then the collating sequence for ORDER BY queries is based on the numeric value of characters (a binary sort that requires fewer system resources). • If the value is a named linguistic sort, sorting is based on the order of the defined linguistic sort. Most (but not all) languages supported by the NLS_LANGUAGE parameter also support a linguistic sort with the same name. Default value: Derived from NLS_LANGUAGE.	BINARY

open_cursors	Specifies the maximum number of open cursors (handles to private SQL areas) a session can have at once. It is important to set the value of OPEN_CURSORS high enough to prevent your application from running out of open cursors. Default value: 50.	Suggested value: 500
session_cached_cursors	Specifies the number of session cursors to cache. Repeated parse calls of the same SQL statement cause the session cursor for that statement to be moved into the session cursor cache. Subsequent parse calls find the cursor in the cache and do not reopen the cursor. Oracle uses a least recently used algorithm to remove entries in the session cursor cache to make room for new entries when needed. This parameter also constrains the size of the PL/SQL cursor cache which PL/SQL uses to avoid having to re-parse as statements are re-executed by a user. Default value: 50	Suggested value: 500
_b_tree_bitmap_plans	This enables use of bitmap access paths for b-tree indexes. Default value: TRUE	Suggested Value: FALSE
Processes	Sets the maximum number of operating system processes that can be connected to Oracle concurrently. The value of this parameter must account for Oracle background processes. SESSIONS parameter is deduced from this value. Default value: 100	Suggested Value: 1500. For a large scale system i.e., for databases with large number of users, the recommended value is: 5000.
sga_target	Setting this parameter to a nonzero value enables Automatic Shared Memory Management. Consider using automatic memory management, both to simplify configuration and to improve performance. Default value: 0	Suggested Value: A minimum of 2gb on small systems and 18gb for large systems.
pga_aggregate_target	Specifies the target aggregate PGA memory available to all server processes attached to the instance. Default value is 0	Suggested Value: A minimum of 1gb on small systems and 8gb for large systems.
Memory_target	MEMORY_TARGET specifies the Oracle system-wide usable memory. The database tunes memory to the MEMORY_TARGET value, reducing or enlarging the SGA and PGA as needed	Consider setting the MEMORY_TARGET to NONE. Set SGA and PGA separately as setting

		MEMORY_TARGET does not allocate sufficient memory to SGA and PGA as needed.
Disk_asynch_io	DISK_ASYNCH_IO controls whether I/O to data files, control files, and log files is asynchronous (that is, whether parallel server processes can overlap I/O requests with CPU processing during table scans Default value: TRUE	If your platform supports asynch IO, leave this parameter to its default value of TRUE, otherwise set it to FALSE.
Filesystemio_options	FILESYSTEMIO_OPTIONS specifies I/O operations for file system files. Default value: None	Suggested value: SETALL
Secure_Files	Specifies how to store LOB objects from tables. Default value: PERMITTED	Suggested value: ALWAYS
Parallel_max_servers	PARALLEL_MAX_SERVERS specifies the maximum number of parallel execution processes and parallel recovery processes for an instance. As demand increases, Oracle Database increases the number of processes from the number created at instance startup up to this value.	Suggested value: Equivalent to NUM_CPU_CORES
Job_queue_processes	JOB_QUEUE_PROCESSES specifies the maximum number of job slaves per instance that can be created for the execution of DBMS_JOB jobs and Oracle Scheduler (DBMS_SCHEDULER) jobs. DBMS_JOB and Oracle Scheduler share the same job coordinator and job slaves, and they are both controlled by the JOB_QUEUE_PROCESSES parameter.	Suggested value: Equivalent to NUM_CPU_CORES

Huge pages:

HugePages is a feature integrated into the Linux kernel 2.6. It is a method to have larger page size that is useful for working with very large memory. HugePages is useful for both 32-bit and 64-bit configurations. HugePage sizes vary from 2MB to 256MB, depending on the kernel version and the hardware architecture. For Oracle Databases, using HugePages reduces the operating system maintenance of page states, and increases Translation Lookaside Buffer (TLB) hit ratio.

Without HugePages, the operating system keeps each 4KB of memory as a page, and when it is allocated to the SGA, then the lifecycle of that page (dirty, free, mapped to a process, and so on) is kept up to date by the operating system kernel.

With HugePages, the operating system page table (virtual memory to physical memory mapping) is smaller, since each page table entry is pointing to pages from 2MB to 256MB. Also, the kernel has fewer pages whose lifecycle must be monitored.

The following are the advantages of using HugePages:

- Increased performance through increased TLB hits.
- Pages are locked in memory and are never swapped out which guarantees that shared memory like SGA remains in RAM.
- Contiguous pages are preallocated and cannot be used for anything else but for System V shared memory (e.g. SGA)
- Less bookkeeping work for the kernel for that part of virtual memory due to larger page sizes

Configuring HugePages on RHEL/ Oracle Linux

Complete the following steps to configure HugePages:

1. Run the following command to display the value of Hugepagesize variable:
2. $ grep Hugepagesize /proc/meminfo
3. Complete the following procedure to create a script that computes recommended values for hugepages configuration for the current shared memory segments:

   Note: Following is an example that may require modifications.

   Please refer to Oracle Support note: Doc ID 401749.1

   Create a text file named hugepages_settings.sh.
   Add the following content in the file:

```
#!/bin/bash
#
# hugepages_settings.sh
#
# Linux bash script to compute values for the
# recommended HugePages/HugeTLB configuration
#
# Note: This script does calculation for all shared memory
# segments available when the script is run, no matter it
# is an Oracle RDBMS shared memory segment or not.
# Check for the kernel version
KERN=`uname -r | awk -F. '{ printf("%d.%d\n",$1,$2); }'`
# Find out the HugePage size
HPG_SZ=`grep Hugepagesize /proc/meminfo | awk {'print $2'}`
# Start from 1 pages to be on the safe side and guarantee 1 free HugePage
NUM_PG=1
# Cumulative number of pages required to handle the running shared memory segments
for SEG_BYTES in `ipcs -m | awk {'print $5'} | grep "[0-9][0-9]*"`
do
  MIN_PG=`echo "$SEG_BYTES/($HPG_SZ*1024)" | bc -q`
  if [ $MIN_PG -gt 0 ]; then
    NUM_PG=`echo "$NUM_PG+$MIN_PG+1" | bc -q`
  fi
done
# Finish with results
case $KERN in
  '2.4') HUGETLB_POOL=`echo "$NUM_PG*$HPG_SZ/1024" | bc -q`;
       echo "Recommended setting: vm.hugetlb_pool = $HUGETLB_POOL" ;;
  '2.6') echo "Recommended setting: vm.nr_hugepages = $NUM_PG" ;;
   *) echo "Unrecognized kernel version $KERN. Exiting." ;;
```

```
esac
# End
Run the following command to change the permission of the file:
$ chmod +x hugepages_settings.sh
```

4. Run the hugepages_settings.sh script to compute the values for hugepages configuration:
5. $ ./hugepages_settings.sh
6. Set the following kernel parameter:
7. # sysctl -w vm.nr_hugepages=value_displayed_in_step 3
8. To make the value of the parameter available for every time you restart the computer, edit the /etc/sysctl.conf file and add the following entry:
9. vm.nr_hugepages=value_displayed_in_step 3
10. Run the following command to check the available hugepages:
11. $ grep Huge /proc/meminfo
12. Restart the instance.
13. Run the following command to check the available hugepages (1 or 2 pages free):
14. $ grep Huge /proc/meminfo

Note: If the setting of the nr_hugepages parameter is not effective, you must restart the server.

Restrictions for HugePages Configurations

Following are the limitations of using HugePages:

- The Automatic Memory Management (AMM) and HugePages are not compatible. With AMM the entire SGA memory is allocated by creating files under /dev/shm. When Oracle Database allocates SGA that way HugePages are not reserved. You must disable AMM on Oracle Database 11g to use HugePages.

SOA DB contention

Most of the SOA workloads generate heavy DML activity in the database and hence likely to experience contention on database objects.

Common popular wait events from SOA DB

Wait event data in AWR report reveals various symptoms of problems that might be impacting performance. The most common wait events that could occur in SOA database are as following:

- DB CPU
- Db file sequential read, db file scattered read
- log file sync
- enq: HW – contention
- enq: TX – index contention
- buffer busy waits
- gc buffer busy acquire, gc buffer busy release (RAC)
- enq: SQ – contention

Some of these events and how to tune them are described below:

log file sync - Tuning redo log:

Most of the times, SOA workload in the database suffers from redo log performance and we see "log file sync" event as one of the top events in AWR. The possible reasons for high "log file sync" waits are as following:

1. LGWR is unable to complete writes fast enough for one of the following reasons:
    a. Disk I/O performance to log files is not good enough.
    b. LGWR is starving for CPU resources.
2. LGWR is unable to post the processes fast enough, due to excessive commits.
3. LGWR is suffering from other database contention such as enqueue waits or latch contention.

Tuning the redo log performance can provide performance improvement for applications running in an Oracle Fusion Middleware environment.

The first step in identifying the root cause is to find and break down LGWR wait events. You can query the wait events for LGWR using its SID as shown below:

```
SQL> SELECT sid, event,  time_waited, time_waited_micro
       FROM v$session_event
       WHERE sid IN
            (SELECT SID FROM v$session WHERE type!='USER' AND
           program LIKE '%LGWR%' )
ORDER BY time_waited;
```

Sizing Redo log:

The guidelines for improving redo log performance are as following:

1. Size online redo logs to control the frequency of log switches and minimize system waits.
2. Optimize the redo log disk to prevent bottlenecks.
3. Determine the optimal sizing of the log_buffer.

The size of the redo log files can influence performance, because the behavior of the database writer and archiver processes depend on the redo log sizes. Keeping the redo log file size too small would cause frequent checkpoints and log file switches and affects system performance. Consider having atleast 3 redo log groups with 2G of size each to start with and monitor the redo log performance periodically and adjust the number of redo log groups and size of each member as appropriate inorder to control the frequency of log switches and minimize system waits. Oracle Enterprise Manager can be used to get sizing advice on the Redo Log Groups. Size the redo log files according to the amount of redo the system generates. A rough guide is to switch logs at most once every 20 minutes. So if your online redo logs (for example) switch once every 5 minutes during peak database activity, to achieve the 20 minute guideline, the logs would each need to be 4 times larger then their current size. (i.e. 20 / 5 = 4)

Place the redo log files on a disk separate from data files to improve I/O performance. SOA database is highly write intensive which generates massive amount of redo per second and per transaction. Sometimes no amount of disk tuning may relieve redo log bottlenecks, because Oracle must push all updates, for all disks, into a single redo location. In SOA database, it is very common to see the foreground wait event "log file sync" as top most events in AWR report with high average wait time. If I/O bandwidth is an issue, doing anything other than improving I/O bandwidth is not useful. One way to relieve redo bottlenecks is to use faster redo storage. It is recommended to use Solid State Disk redo log files. SSD has greater bandwidth than platter disk.

Tuning Redo Log buffer:

SOA applications insert, modify and delete large volumes of data and most of these operations are committed in a row-by-row fashion rather than in batch mode. Because of too frequent committing of transactions there is a significant overhead on the redo performance. Hence it is important to size log_buffer optimally.

The statistic REDO BUFFER ALLOCATION RETRIES from AWR report and/or from V$ views reflects the number of times a user process waits for space in the redo log buffer. This statistic can be queried through the dynamic performance view V$SYSSTAT.
Use the following query to monitor these statistics over a period while your application is running:

```
SELECT NAME, VALUE
FROM V$SYSSTAT
WHERE NAME = 'redo buffer allocation retries';
```

The value of redo buffer allocation retries should be near zero over an interval. If this value increments consistently, then processes have had to wait for space in the redo log buffer. The wait can be caused by the log buffer being too small or by check pointing. Increase the size of the redo log buffer, if necessary, by changing the value of the initialization parameter LOG_BUFFER. The value of this parameter is expressed in bytes. Alternatively, improve the check pointing or archiving process. Another data source is to check whether the log buffer space wait event is not a significant factor in the wait time for the instance; if not, the log buffer size is most likely adequate. Use caution while increasing log_buffer setting, because excessive redo size can also cause high "log file sync" waits and hurts performance.

A good starting rule of thumb for a write intensive workload is to configure the log buffer to 100mb.

Tuning LGWR process:

In most of the SOA workloads, commit rate is very high, and decreasing commits is not an option, as a result high "log file sync" waits are a common event listed in the top 5 in AWR report. Follow the guidelines illustrated above to address high "log file sync", if the redo log performance still does not improve, finally try to increase priority of LGWR (using nice) or increasing priority class of LGWR to RT might provide some benefit.

Smart Flash Logging:

If the database is on ExaData machine, it is recommended to uptake a minimum of BP11 to take advantage of the Smart Log Feature.

An additional feature implemented in Exadata Storage software 11.2.2.4.2 (and database version 11.2.0.2 + BP11) is called Exadata Smart Flash Logging. With this feature 512MB of flash storage is reserved for redo writes and the database's log writer (LGRW) process adopts a different pattern of behavior. In a system which does not use this feature LGWR writes in parallel to multiplexed copies of the redo logs and then waits for all writes to complete. The result of this is that the time taken to perform these writes (indicated by the Oracle wait interface statistics log file parallel write) is the time taken for the slowest disk to complete the write. With Exadata Smart Flash Logging the redo log files remain on disk but the additional reserved 512MB of space is created on flash storage. When issuing a write call, LGWR write to the redo logs on disk as usual but will additionally make another parallel write to the flash area. LGWR then waits for whichever of these writes completes first to post it, after which it continues without waiting for the other; in this method the redo write time is now that of the fastest media – flash or disk.

enq:HW contention

enq: HW contention for Busy LOB segments: The HW High Water enqueue contention occurs when competing processes are inserting into the same table and are trying to increase the high water mark of a table simultaneously.

In SOA database, this issue is experienced by tables that have LOB columns. For e.g., CUBE_SCOPE, XML_DOCUMENT, AUDIT_DETAILS etc., Under heavy load the LOB segments in these tables experience contention which is evident from AWR report through the wait event "enq: HW contention". The default storage for LOBs in Oracle 11g database is BASICFILE hence the problem of HW contention arises. Frequent allocation of extents, or reclaiming chunks may be causing contention for the LOB segments high water mark. HW enqueue contention can occur for LOB segments which are ASSM managed as space allocation only acquires one block at a time.

This type of contention can affect performance significantly and the contention can be eliminated using SecureFiles feature of Oracle database Release 11g. SecureFiles are a new LOB storage architecture which provides performance benefits over traditional BasicFile LOBs.

Migration of BasicFiles to SecureFiles can be done using one of the following methods:

1. Set the database parameter SECURE_FILES = ALWAYS – This method is applicable for new installations prior to creating SOA tables using RCU. Once this parameter is set at the instance level, any new LOB segments created will use SecureFiles automatically.

2. Migrate LOBs to SecureFiles: This method is applicable for installations that already have SOA tables created in them. In such cases, LOB segments from tables in a SOA database experiencing enq: HW contention can be migrated to SecureFiles. This can be achieved using online redefinition method with very little downtime. The pl/sql package DBMS_ONLINE_REDEFINITION facilitates this migration without loosing any data, constraints, indexes. Please visit Oracle Support Note: 1498415.1 to get a sample script for migrating LOBs to SecureFiles.

3. Set the database event 44951.

   ALTER SYSTEM SET EVENT='44951 TRACE NAME CONTEXT FOREVER, LEVEL 1024? scope=spfile;

   If a SOA installation is using Oracle version older than 11g then, this database event can be set to avoid "enq:HW contention" on LOB segments.

The following table lists the LOB columns that are likely to suffer from the "enq:HW – contention" under heavy load and it is highly recommended to move these LOBs to SecureFiles. Thorough analysis of AWR, ADDM reports is needed to know which LOB objects are suffering from "enq:HW – contention". The below table may be used as a reference.

Table Name	Column Name	Recommended LOB storage Attributes
ATTACHMENT	ATTACHMENT	COMPRESS CACHE
AUDIT_DETAILS	BIN	COMPRESS CACHE
CUBE_SCOPE	SCOPE_BIN	COMPRESS CACHE

Index Contention

In most of the SOA scenarios, multiple database sessions insert thousands of rows into SOA tables and many of the index keys are monotonically increasing, particularly the primary key indexes. Indexes of a B*Tree structure will insert these keys targeting only a few database blocks which can become very hot across a Real Application Cluster(RAC). This issue is seen in the AWR report as high "Buffer Busy waits".

Another aspect of BTREE indexes that leads to contention on indexes being inserted into is that, when a transaction inserting a row in an index has to wait for the end of an index block split being done by another transaction the session would wait on event enq: TX - index contention. In RAC environment, the contention on index is shown as "gc buffer busy acquire" & "gc buffer busy release" wait events. When high number of concurrent inserts lead to excessive index block splits, it hurts the performance.

Solution is to reorganize the index in a way to avoid the contention or hot spots. To distribute the index keys randomly across many database blocks the indexes can be Global Hash partitioned.

Hash partitioning has proven the best tuning method to address index contention. The following table lists the indexes from SOA tables that are likely to suffer from the index contention under heavy load and the table gives recommendations on how to partition such indexes. Thorough analysis of AWR, ADDM reports is needed to know what indexes need to be partitioned, the below table may be used as a reference.

Index	PARTITIONING KEY
COMPOSITE_INSTANCE_CREATED	GLOBAL PARTITION BY RANGE (CREATED_TIME) and then SUBPARTITION BY HASH(COMPOSITE_DN)
BRDECISIONINSTANCE_INDX3	GLOBAL PARTITION BY RANGE(CREATION_TIME) and then SUBPARTITION BY HASH(STATE)
MEDIATOR_INSTANCE_INDEX2	GLOBAL PARTITION BY RANGE(CREATED_TIME) and then SUBPARTITION by LIST(COMPONENT_STATE)
MEDIATOR_INSTANCE_INDEX5	GLOBAL PARTITION BY HASH (COMPLETED_CASE_NUM,CASE_NUM,COMPONENT_NAME)
MEDIATOR_INSTANCE_INDEX6	GLOBAL PARTITION BY HASH (COMPONENT_STATE, COMPONENT_NAME,CREATED_TIME)
MEDIATOR_INSTANCE_INDEX1	GLOBAL PARTITION BY HASH (ID)
MEDIATOR_INSTANCE_INDEX3	GLOBAL PARTITION BY HASH (COMPONENT_NAME,COMPONENT_STATE,CREATED_TIME)
MEDIATOR_CASE_INSTANCE	GLOBAL PARTITION BY HASH(ID)
MEDIATOR_CASE_INSTANCE_INDEX2	GLOBAL PARTITION BY HASH(INSTANCE_ID)
MEDIATOR_CASE_DETAIL_INDEX1	GLOBAL PARTITION BY HASH(INSTANCE_ID)
REFERENCE_INSTANCE_CO_ID	GLOBAL PARTITION BY HASH(PROTOCOL_CORRELATION_ID)
DOC_DLV_MSG_GUID_INDEX	GLOBAL PARTITION BY HASH(MESSAGE_GUID)

CUBE_SCOPE Primary Key INDEX	GLOBAL PARTITION BY HASH(CIKEY)
XML_DOCUMENT Primary Key INDEX	GLOBAL PARTITION BY HASH (DOCUMENT_ID)
CI_ECID	GLOBAL PARTITION BY HASH(ECID)
AC_PK	GLOBAL PARTITION BY HASH(CIKEY)
CI_CREATION_DATE	GLOBAL PARTITION BY RANGECREATION_DATE)
TABLE AUDIT_COUNTER	PARTITION BY HASH(CIKEY)
WI_STRANDED	GLOBAL PARTITION BY RANGE(MODIFY_DATE)
DLV_MESSAGE_CIKEY	GLOBAL PARTITION BY HASH(CIKEY)
DOC_DLV_MSG_GUID_INDEX	GLOBAL PARTITION BY HASH(MESSAGE_GUID)
HEADERS_PROPERTIES_PK	GLOBAL PARTITION BY HASH(MESSAGE_GUID)
DLV_MESSAGE_PK	GLOBAL PARTITION BY HASH(MESSAGE_GUID)
DLV_MESSAGE_DATE_ECID	GLOBAL PARTITION BY RANGE(RECEIVE_DATE) and then SUBPARTITION by HASH(ECID)
DOC_DLV_MSG_ID_INDEX	GLOBAL PARTITION BY HASH(DOCUMENT_ID)
DLV_MESSAGE Primary Key Index	GLOBAL PARTITION BY HASH (MESSAGE_GUID)
DM_CONVERSATION	GLOBAL PARTITION BY HASH(CONV_ID);
CONV_ID_STATE_DLV_TYPE	GLOBAL PARTITION BY HASH(CONV_ID)
WI_STRANDED	GLOBAL PARTITION BY RANGE (MODIFY_DATE)

Sequence contention

The following table lists the sequences that are likely to suffer from the contention under heavy load and the table gives recommendations on how to optimize those sequences. Thorough analysis of AWR, ADDM reports is needed to know if any sequences are suffering from contention. The below table may be used as a reference.

Sequence Name	Recommendation
SEQ_ULGYF_QHGPHCDX_ZBG5ARQ__	Increase the cache value to 2000
SEQ_Kz5RKYJvLhWspE3N_p13ew__	Increase the cache value to 2000

Purging

Installations of SOA that accumulate a lot of data should implement purging strategy to clean up redundant data and help performance of SQL queries and also to save disk space.

The need for aggressive and continuous purging is a key aspect to control performance and disk space in SOA.

Please refer to the following SOA purging white paper for more details on implementation.

SOA 11G Database Growth Management Strategy

Space Reclamation

SOA installations that implement frequent purging of unwanted data from SOA tables are more likely to experience disk space issues. The purge scripts delete rows from database segments (tables and indexes) releasing space within the data blocks for reuse but may also cause fragmentation with some space too small for reuse. Even with ASSM and locally managed table spaces it is possible to notice that the space is not released as soon as rows are deleted, particularly when the tables contain LOB columns. Hence it is recommended to shrink tables & LOB columns to reclaim space manually.

The Shrink operation consolidates free space below the high water mark and compact the segment, after which it then moves the high water mark and de-allocated the space above the high water mark.
The commands that can be used to reclaim space manually are as following:

```
ALTER TABLE CUBE_SCOPE ENABLE ROW MOVEMENT;
ALTER TABLE CUBE_SCOPE SHRINK SPACE;
ALTER TABLE CUBE_SCOPE MODIFY LOB (SCOPE_BIN) (SHRINK SPACE);
ALTER TABLE CUBE_SCOPE DISABLE ROW MOVEMENT;
```

The Segment Advisor can be used to identify segments that would benefit from online segment shrink. However after constant purging most SOA segments should be candidates for online segment shrink operations. (Please refer to the Oracle Database Administration guide for more information on Segment Advisor)

Optimizer Statistics Collection

Optimizer statistics describe details about the database and the objects in the database. The query optimizer uses these statistics to choose the best execution plan for each SQL statement.

Optimizer statistics include the following:

- Table statistics
  - Number of rows, Number of blocks, Average row length
- Column statistics
  - Number of distinct values (NDV) in column, Number of nulls in column
  - Data distribution (histogram),Extended statistics
- Index statistics
  - Number of leaf blocks, Levels, Clustering factor
- System statistics
  - I/O performance and utilization, CPU performance and utilization

Automatic Statistics Gathering

Because objects in a database can change constantly, you must update statistics regularly so that they accurately describe these objects. For all SOA databases it is recommended to use the Automatic Statistics Collection that is enabled by default in Oracle 11g database and this job runs every night.

Manual Statistics Gathering

Automatic optimizer statistics collection is sufficient for most of the database objects, but in a database that is close to going live or for tables that are modified/purged significantly, manual statistic gathering is needed. Also, for SOA databases that implement purging of stale data on regular basis, it is recommended to collect stats manually right after purging has completed.

In these cases, use the DBMS_STATS.GATHER_TABLE_STATS procedure.

MDS DB: Collecting statistics for optimizing the MDS database repository performance

Ensure auto-stats collection is enabled.

In most cases, the first 32 characters of PATH_FULLNAME in the MDS_PATHS table are the same. You can prevent the database putting them in the same section of the histogram by doing the following:

Drop the histogram for PATH_FULLNAME column by executing the following as system.

```
execute dbms_stats.delete_column_stats(ownname=>'mdsSchemaOwner', tabname=>'MDS_PATHS',
colname=>'PATH_FULLNAME', col_stat_type=> 'HISTOGRAM');
```

Set table preferences to exclude collecting histogram for the PATH_FULLNAME column.

```
execute dbms_stats.set_table_prefs(mdsSchemaOwner, 'MDS_PATHS', 'METHOD_OPT', 'FOR
COLUMNS SIZE 1 PATH_FULLNAME');
```

Fixed Object Statistics

The automatic statistics gathering job does not gather fixed object statistics. The Optimizer uses predefined default values for the statistics if they are missing. These defaults may not be representative and could potentially lead to a suboptimal execution plan, which could cause severe performance problems in your system.

You can collect statistics on fixed objects using DBMS_STATS.GATHER_FIXED_OBJECTS_STATS procedure. Because of the transient nature of the x$ tables it is import that you gather fixed object statistics when there is a representative workload on the system.

It is recommended that you re-gather fixed object statistics if you do a major database or application upgrade, implement a new module, or make changes to the database configuration. For example if you increase the SGA size then all of the x$ tables that contain information about the buffer cache and shared pool may change significantly, such as x$ tables used in v$buffer_pool or v$shared_pool_advice.

System Statistics

System statistics allow the optimizer to consider a system's I/O and CPU performance and utilization. For each candidate plan, the CBO optimizer computes estimates for I/O and CPU costs. It is important to know the system characteristics to pick the most efficient plan with the optimal proportion between I/O and CPU cost. System CPU and I/O characteristics depend on many factors and do not stay constant all the time. The gathered statistics are:

- single block readtime in ms
- multiblock readtime in ms
- cpu speed in mhz
- average multiblock_read_count in number of blocks

These statistics allow the optimizer to scale costs so that there are more accurate proportional to elapsed time.

System statistics should be gathered when a new hardware (CPU, disk) is added and/or when the system load changes

System statistics are not gathered by automatic statistics gathering job. DBAs can collect system statistics when necessary. The frequency of collection should be dependent on the nature of the system itself. If the systems hardware and data load is static, then it would be unnecessary to continually re-collect system statistics. However if application load and machine load is highly variable, then a more frequent approach is needed. Also note that if the system load varies at different times of the day, then different system stats may be used to reflect these loadings

Steps for gathering system statistics:

-- Start gathering system statistics before you start running the workload

```
EXECUTE dbms_stats.gather_system_stats('Start');
```

-- Run a typical application workload

-- Stop gathering system statistics after the workload has finished.

```
EXECUTE dbms_stats.gather_system_stats('Stop');
```

SQL Plan Baseline:

Sometimes, SQL statements suffer from change in execution plan all of a sudden. We may notice that, one or more SQLs that were performing efficiently started to experience poor performance even without any database changes like index changes, database parameter changes, patches etc., This is all because CBO picked up a new plan that is not optimal hence SQL performance suffers. You may find this from AWR data or by looking at recommendations given by SQL Tuning advisor available via Enterprise Manager.

The change in execution plan can occur due to stale statistics, changes in data volume or bind peeking and as a result the SQL performance may degrade. In such situations, one can try fixing the problem by recollecting optimizer statistics. If the change in plan persists still, SQL plan baseline can be created to solve the performance of the sql statement.

The goal of SQL plan baselines is to preserve the performance of corresponding SQL statements, regardless of changes in the database. A SQL plan baseline contains one or more accepted plans, each of

which contains the following information: Set of hints, Plan hash value, and Plan-related information. Instructions for creating SQL plan baseline as are following:

1. Using AWR data (or) SQL Tuning advisor identify the slow sql that suffers from change in the execution plan.

2. Create a SQL tuning task

```
DECLARE
  l_sql_tune_task_id  VARCHAR2(100);
BEGIN
  l_sql_tune_task_id := DBMS_SQLTUNE.create_tuning_task (
                                        begin_snap  => 3999,
                                        end_snap    => 4195,
                                        sql_id      => 'b4440hp0hmkq0',
                                        scope       => DBMS_SQLTUNE.scope_comprehensive,
                                        time_limit  => 60,
                                        task_name   => 'b4440hp0hmkq0_AWR_tuning_task',
                                        description => 'Tuning task for b4440hp0hmkq0');
  DBMS_OUTPUT.put_line('l_sql_tune_task_id: ' || l_sql_tune_task_id);
END;
/
```

-- In the above step, change the SNAP IDs, SQL ID, and Task Name as per the target database

3. Execute the SQL Tuning task.

```
EXEC DBMS_SQLTUNE.execute_tuning_task(task_name => b4440hp0hmkq0_AWR_tuning_task');
```

4. Generate the report from SQL tuning task.

```
SELECT task_name, status
FROM dba_advisor_log
WHERE task_name='b4440hp0hmkq0_AWR_tuning_task';

set long 1000000

SELECT DBMS_SQLTUNE.report_tuning_task('b4440hp0hmkq0_AWR_tuning_task')
AS recommendations FROM dual;
```

5. Create SQL Plan baseline

```
execute dbms_sqltune.create_sql_plan_baseline(object_id=> 1,task_name
=>'b4440hp0hmkq0_AWR_tuning_task', owner_name => 'SYSTEM',plan_hash_value => 613666522);
```

-- In the above step, change the object_id & plan hash value based on the output of the report from tuning task.

6. Drop the sql tuning task.

```
BEGIN
  DBMS_SQLTUNE.drop_tuning_task (task_name => 'b4440hp0hmkq0_AWR_tuning_task');
END;
/
```

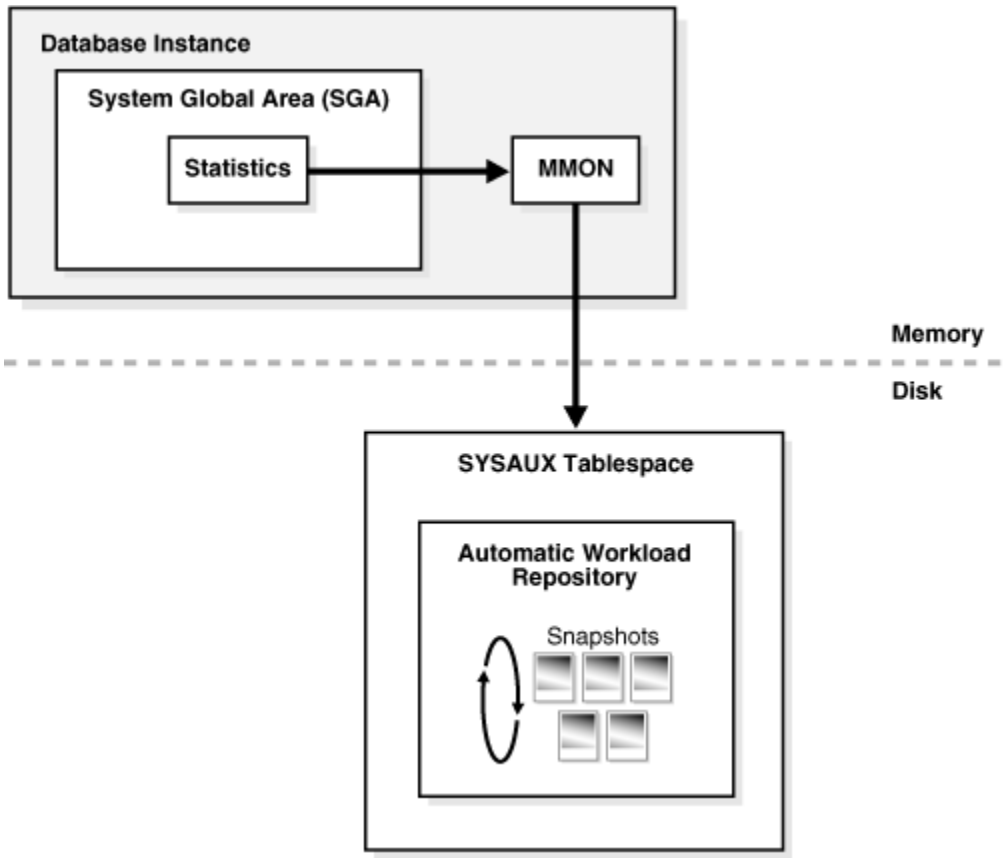
Please see more information @
https://blogs.oracle.com/optimizer/entry/sql_plan_management_part_1_of_4_creating_sql_plan_baselines

Appendix

Oracle provides Automatic Workload Repository (AWR) and Automatic Database Diagnostic Monitor (ADDM) to gather and analyze database performance statistics.

AWR, ADDM, ASH reports

- A built-in repository in every Oracle database
- Statistics stored in AWR are foundation for performance tuning.



AWR can be configured to automatically collect performance statistical snapshots at regular intervals, or can be manually triggered. The ADDM utility can then be used to analyze the statistics between two snapshot intervals, to produce a performance summary report that includes:

- CPU bottlenecks
- Undersized Memory Structures
- I/O capacity issues
- High load SQL statements
- High load PL/SQL execution and compilation, and high-load Java usage
- Oracle RAC specific issues
- Sub-optimal use of Oracle Database by the application
- Database configuration issues
- Concurrency issues, Hot objects

The AWR and ADDM utilities can be executed manually or via EM Database console. (Please refer to Oracle Database Administrator Guide for information about using Oracle Enterprise Manager.).

AWR Reports

Taking AWR Snapshots

1. From SQLPLUS session, connect to the database as a user with DBA privileges.

2. Take the begin AWR snapshot before the test run is started using the following command:
SQL> SELECT DBMS_WORKLOAD_REPOSITORY.Create_Snapshot FROM DUAL;

```
CREATE_SNAPSHOT
-------------------------------
        1904
```

3. Complete your test run

4. Take the end AWR snapshot after the test run is completed using the following command:
SQL> SELECT DBMS_WORKLOAD_REPOSITORY.Create_Snapshot FROM DUAL;

```
CREATE_SNAPSHOT
-------------------------------
        1905
```

ADDM Reports

- Self-diagnostic advisor.
- ADDM identifies areas of Oracle Database consuming the most time.
  - Lock contention
  - Excessive parsing
  - I/O capacity
  - Incorrect sizing of SGA, PGA
  - And many more……
- Recommends solutions and quantifies expected performance benefits.
- Recommends changes to hardware, database configuration, database schema, or applications

ASH Reports

- Historical view of active sessions. – Sampled every second
- Enables “after the fact” analysis.
- Drilldown to a more granular period .
- P1, P2, P3 values
- List details of only a session, SQL, Wait class service, module over a period of a time.
- Determines blocking session, hot segment.

- Generating AWR, ADDM, ASH Reports:

    1. Generate AWR report using the following command:

SQL> @?/rdbms/admin/awrrpt.sql

2. Generate ADDM report:

SQL> @?/rdbms/admin/addmrpt.sql

3. Generate ASH report:

SQL> @?/rdbms/admin/ashrpt.sql

4. Generate AWR Diff report:

SQL> @?/rdbms/admin/awrddrpt.sql

References

Oracle® Database Performance Tuning Guide 11g Release 2 (11.2)

Oracle® Database VLDB and Partitioning Guide 11g Release 2 (11.2)

SOA 11g Database Performance
February 2013
Author:
Srinivas Kasam

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

www.oracle.com