Oracle Maximum
Availability Architecture

An Oracle 11g Fusion Middleware SOA White Paper
January 2012

SOA 11G Database
Growth Management Strategy

Disclaimer

Executive Overview 1
Introduction 1
1. SOA Database Profile 2
  1.1. Inflow of data 3
  1.2. Retention Policy 4
  1.3. Outflow of data 4
  1.4. Long Running Composites and Table Partitioning 5
2. SOA Database Growth Management Strategy 7
  2.1. Database Growth Management Challenges 7
  2.2. Quality Assurance Testing 9
  2.3. Recommended Strategies. 11
  2.3.1. Recommendations for Large Database profiles 11
  2.3.2. Recommendations for Medium Database profiles 11
  2.3.3. Recommendations for Small Database profiles 11
3. Troubleshooting. 13
  3.1. Parallel Purge 13
  3.1.1. Degree of Parallel 13
  3.1.2. Parsing Composites to Delete 14
  3.1.3. Parallel Query Slaves 14
  3.1.4. Debug and Tracing 15
  3.2. SOA Table Partitioning. 17
  3.2.1. Referential Integrity and Equi-Partitioning 17
  3.2.2. Composite Range-Hash partitions 18
  3.2.3. Interval Partitioning. 18
  3.2.4. Global Hash Indexes. 18
  3.2.5. Partition Pruning 19
  3.2.5. Purging Partitions 19
  3.3. Reducing Audit levels 19
Conclusion 20
References 21

Appendix A. Monitoring the SOA Installation 22
A.1. Monitor Space Usage.. 22
A.1.1. Determine the growth trend of the components. 22
A.1.2. Determine the growth trend of the SOA Schema............... 23
A.1.3. Determine the largest segments.. 23
A.1.4. Determine the growth trend of tables and indexes............. 24
A.1.5. Estimate Table size.. 26
A.1.6. Estimating Index size. 29
A.1.7. Monitor unused Indexes. 30
A.2. Monitor Hardware and Database... 30
A.2.1. Hardware - OS Watcher 30
A.2.2. Database – AWR / ADDM 31
A.2.3. Disk I/O - Orion 31
Appendix B. Space Management 33
B.1. Concepts... 33
B.2. Reclaiming Segment and Data file Space 35
B.2.1. Online Segment Shrink.. 35
B.2.2. De-allocate unused space 37
B.2.3. Index Rebuild and Coalesce.. 37
B.2.4. Table Partitions 38
B.2.5. Secure file LOB 39
B.2.5.1. Secure file requirements... 39
B.2.5.2. Secure files conversion. 40
B.2.5.3. Secure file migration... 41
B.2.6. Other Techniques.. 41
B.3. Data files resizing.. 42
Appendix C: Purging / Partitioning Tooling Details........................... 43
Purge / Partitioning Tooling 43
C.1. Single Thread Purge 43
C.2. Parallel Threaded Purge 46

C.3. SOA Table Partitioning.. 48
C.4. Row Movement Scripts 51

Executive Overview

This whitepaper has been written to highlight the need to implement an appropriate strategy to manage the growth the of SOA 11g database. The advice presented should facilitate better dialog between SOA and Database administrators when planning database and host requirements.

Introduction

An Oracle SOA 11G installation presents a few challenges for administrators and one of these is managing the growth of the SOA database. The importance of managing the database can be under emphasized leading to issues post go-live. This paper will help administrators determine an appropriate strategy and highlights the need for capacity planning, testing and monitoring.

The recommended strategies are based on the profile or the predicted size of the SOA installation. The tools and techniques that implement the strategies are reasonably straight forward which helps to simplify the recommendations. This does however restrict administrators to a set of tools which should be implemented as deigned.

Testing and monitoring are required to determine the effectiveness and resource requirements of a growth management strategy. Thorough testing will ensure that the tools, the database and hardware resources all come together to meet the needs of the current and future database growth estimates.

The intended audience for this paper is SOA 11g and Database Administrators.

1. SOA Database Profile

This section will detail the steps to determine the profile or size of the SOA database in order to determine the optimal growth management strategy (refer Strategy section).

The calculations detailed below to approximate disk space usage are not a replacement for conducting a thorough space capacity plan. The estimates are however sufficient to draw conclusions on the appropriate growth management strategy while highlighting the need for disk space planning.

The input to determine the profile:

- Inflow (refer 1.1).
- Retention Policy (refer 1.2).

Additional considerations:

  - Outflow (refer 1.3).
  - Long Running Composites and Table Partitioning (refer 1.4).

The output that determines the profile:

- Composites space persisted daily (refer 1.1.3).
- Minimum retention of space (refer 1.2.1).

Note: This section will make constant reference to Appendix A which details the space calculations instructions.

Table 1; provide the profile of small, medium and large installations based on "composite space persisted daily" and "Minimum retention space". These two metrics are related in an "and/or" condition as retention policy may not retain more than a few days of data but composite inflow might be high.

TABLE 1 SOA DATBASE PROFILES

DB PROFILE	DESCRIPTION
Small	Composites space persisted daily: < 10GB Minimum retention of space: < 100GB
Medium	Composites space persisted daily: Between 10-30GB Minimum retention of space: Between 100-300GB

Large	Composites space persisted daily: Over 30GB Minimum retention of space: Over 300GB

1.1. Inflow of data

The rate of composite inflow and space usage is best understood as an average derived after many days of load testing. This will allow for allocated space (segment extents) to be better utilized. To understand inflow the following data points will be investigated:

- Number of composite produced daily.
- Disk space used by each composite.
- Composites space persisted daily.
- The space distribution of the SOA Segments.

1.1.1. Number of composite produced daily (refer A.1.1)

Daily-inflow-composite = (Composite Total / Period)

E.g.: Calculating the average numbers of composites produced daily; based on 5 days of load testing with a total composite count of 100,000: (20,000 = (100,000 / 5))

1.1.2. Disk space used by each composite (refer A.1.2)

Inflow-space-per-composite = (SOA Schema Size /Composite Total)

E.g.: Calculating the average disk space used by each composite; with a total of 100,000 composites and a SOA schema size of around 200GB: (2MB = (200GB / 100,000))

1.1.3. Composites space persisted daily

Daily-inflow-composite-space =
(Daily-inflow-composite * Inflow-space-per-composite)

E.g.: Calculate the average disk space used by composites daily; with 20,000 composites daily of 2MB each: (40GB = (20,000 * 2MB))

Alternatively:

Daily-inflow-composite-space = (SOA Schema Size / Period):

1.1.4. The space distribution of the SOA Segments.

The shape of the SOA schema should be analyzed so that the distribution of segments space (tables, indexes, lobs) is understood and segments which may be of concern are identified:

- Determine the growth trend of the components (refer A.1.1)

    - The collection of component statistics needs to be gathered after each day of load testing, so that growth averages can be determined.
- Determine the growth trend of the SOA Schema (refer A.1.2)
    - The collection of schema sizing statistics needs to be gathered after each day of load testing, so that growth averages can be determined.
- Determine the largest segments (refer A.1.3)
- Determine the growth trend of tables and indexes. (refer A.1.4)
    - The growth statistics for table and indexes are automatically gathered via the Automatic Workload Repository (AWR).

1.2. Retention Policy

It is important to give proper consideration for how long composites will be retained in the database, as this will affect the size of the SOA schema and the performance of the purge scripts. Factors that drive the retention policy are legal requirements, line of business requirements or overall company policy on retention of data. The longer the retention policy, the greater the volume of data that needs to stored and correspondingly, higher disk capacity requirements.

1.2.1. To determine the minimum retained disk space:

Min-space-retain =
Daily-inflow-composite-space (refer 1.1.3) * Retention Period in days.

E.g.: With 40GB of composites space persisted daily and a retention period of 10 days: (400GB = (40GB * 10))

1.2.2. To determine the minimum number of retained composites:

Min-composites-retain =
Daily-inflow-composite (refer 1.1.1) * Retention Period in days.

E.g.: With 20,000 composites created daily and a retention period of 10 days: (200,000 = (20,000 * 10))

We can state that 200,000 composites retain on average 400GB of data based on a 10 day retention policy.

Note: The above defines an approximation on the minimums disk space usage. The actual minimum disk space usage will be more accurately determined via Quality Assurance testing.

1.3. Outflow of data

The outflow of composites is a measure on the number of composites that are deleted. It is not a factor when determining the database profile but it is important metric when evaluating the effectiveness of the growth management strategy.

The goal of measuring outflow is to ensure that the inflow can be deleted, to ensure a steady state for space usage hopefully just above the minimum retained disk space usage (refer 1.2.1).

Firstly there are two cycles that need to be defined:

- The purge cycle, refers to a period which may involve multiple executions of the purge scripts.
- The maintenance cycle, refers to the number of days required to perform all space management operations (refer Appendix B.1).

The appropriate growth management strategy might include both purging and partitioning thus outflow should not be measure until eligible partitions have been dropped. The dropping of partitions is a maintenance operation, so it makes little sense to measure outflow metrics until the maintenance cycle has finished.

The following metrics can be captured **before and after** the completion of the maintenance cycle to understand the effectiveness of the strategy and to reevaluate space distribution:

- Determine the growth trend of the components (refer A.1.1)
- Determine the growth trend of the SOA Schema (refer A.1.2)
- Determine the largest segments (refer A.1.3)
- Determine the growth trend of tables and indexes. (refer A.1.4)

Note: If a steady state has been reached then the difference between the before and after metrics for components and schema growth should be near zero or negative. Otherwise the purge strategy may not be coping or the inflow of data may have increased.

1.4. Long Running Composites and Table Partitioning

Long running composites are composites which remain open beyond the retention period. These composite do not have significant implications for the purge scripts but their impact is felt when table partitioning is implemented. (The strategy section will recommend table partitioning is considered for medium and large installations.)

The partitioned tables are (should be) excluded from the purge scripts as their space will be reclaimed via the database "alter table … drop partition" command. Long running composites will remain open beyond the desired retention period preventing the partition from being dropped and the space from being reclaimed. Therefore when estimating the size of the partitioned tables the retention period that is used should equal the longest running composite.

Retention period = Longest Running Composite.

Understanding space implications of long running transactions and the size of the partitioned tables is best understood through QA testing as conveyed in the Strategy section. However the following can be used to estimate the space used by each table that will be partitioned:

1.4.1. For each table to partition:
It is best to load the tables over a period of a few days to produce reasonable averages.

1.4.1.2. Determine the average number of rows generated daily for each table

(Total rows / period)

1.4.1.3. Estimate number of rows based on longest running composite:

((Total rows / period) * Longest running composite in days)

1.4.1.4. Estimate space usage for the table and index based on rows calculations (refer appendix A.1.5 & A.1.6)

Note: As of Oracle FMW SOA Suite PS5 a new set of scripts called “row migration” have been provided. These scripts can move long running composites from one partition to another thus facilitating the removal of the partition. However it is best practice to plan for the space requirements of the partition tables and not rely on the row migration scripts.

2. SOA Database Growth Management Strategy

This section will provide recommendations on the appropriate tools to manage the growth of the SOA database. The provided advice is for new installations or ones looking to revise their current strategy that may lack procedure like monitoring and space management (The Troubleshooting section should also be reviewed for additional advice.).

This section is broken into the following.

- SOA Database Growth Management Challenges
- Quality Assurance Testing
- Recommended strategies
    - Recommendations for Large Profile Installations
    - Recommendations for Medium Profile Installations
    - Recommendations for Small Profile Installations
    - Maintenance

It is advisable to read Appendix A prior to this section, in order to become familiar with the SOA purging and partitioning tools.

2.1. Database Growth Management Challenges

The challenges below provided some background by detailing situations which have contributed to the recommended strategies.

2.1.1. SOA tables growing excessively due to an ineffective management strategy.

When an ineffective purging strategy has been implemented the SOA suite tables may have grown very large leading to an urgent need to reclaim space. The larger the tables the harder it will be to delete rows and reclaim space.

The performance of the parallel purge relies on factors like; CPU resources and the speed of disk I/O but very large tables have proven to be challenging. The challenge is due to the amount of data that needs to be parsed in order to determine composites that can be deleted. This parsing can monopolize the elapse time of the entire purge script (refer 3.1.3).

This situation highlights the need to constantly monitor the effectiveness of the growth management strategy and to take corrective actions as soon as possible before the tables become very large.

2.1.2. Parallel Purge is tuned but unable to cope with the inflow.

To help the performance of the parallel purge, the tables with long elapse times can be ranged partitioned and excluded from the purge script. This however raises concerns of downtime if table partitioning is deemed necessary post go-live. While tables can be partitioned online via the redefinition packages, this operation may not complete in a reasonable time when presented with a very large table.

As of SOA PS5 the granularity of tables that can be partitioned has improved. It is now possible to partition high volume tables with only their master table, this should make partitioning more attractive and feasible (refer C.3.3.2).

2.1.3. Table partitions cannot be dropped due to long running composites.

Long running composites are composites which remain open beyond the retention period. This has implications for SOA table partitions as they cannot be dropped until all composites they house are closed. Only a few open composites can prevent a partition from being dropped and space reclaimed. This situation has given rise to the new "row migration scripts" which are available in SOA PS5 (refer C.4)

The recommendation is to plan for sufficient disk space to accommodate partitioned tables which included the retention period of the longest running composites. The row migration scripts are recommended for corrective actions when there is an urgent need to reclaim space.

The movement of composites data by the row migration scripts raises the following concerns::

- The scripts were written with an assumption that around 5% of the data in a partition would be moved to avoid performance concerns with the script.
  - To maintain the requirements of equi-partitioning (refer 3.2.1), would require the movement of 5% of the rows across many table.
  - The arbitrary figure of 5% does not convey the size of the partition which will depend on inflow rate and partition period; daily, weekly, monthly
- Repeated migration of long running composites to either the latest partition or to a partition specifically created to pools active composites, may lead to an accumulation.
  - A given partition may become very large and require purging which has other implications (refer 3.2.5).

2.2. Quality Assurance Testing

Testing the strategies is essential so that the tools and techniques are practiced and understood.

The hardware resources available for QA testing are unlikely to be identical to production, so administrators need to make conservative predictions when projecting results onto the production host.

QA Testing Plan

2.2.1. Review Metalink Support note 358.1

Review Support Doc ID 358.1 "Maintenance and Administration Advisor: Oracle Fusion Middleware (FMW) SOA 11g Infrastructure Database"

2.2.2. Production Setting.

Ensure that audit settings are appropriate for production (refer 3.3).

2.2.3. SOA Schema - Test Bed

Though difficult it is important to create a test bed that is comparable to production. The performance of the purge scripts are adversely affected by the amount of data that needs parsing, so a well sized test bed is important to understand purge performance (refer 3.1.3).

The workload mix which populates the schema must resemble production in payload size and number of composite invocations. The data must simulate many days of loading, so that space averages and size estimates can be better determined.

2.2.3.1. Measure inflow and space estimations.

During the loading of data time should be taken to determine the space used per table and respective space distribution (Refer to Section 1.1for calculations on inflow):

- Determine the growth trend of the components (refer A.1.1)
- Determine the growth trend of the SOA Schema (refer A.1.2)

Collects metrics with partitioning in mind:

- Determine the largest segments (refer A.1.3)
- Determine the growth trend of tables and indexes. (refer A.1.4)
- Estimate table size (refer A.1.5)
- Estimate index size (refer A.1.6)

2.2.3.2. Base point backup of test bed.

2.2.4. Execute parallel or single threaded script and reclaim space.

Firstly there are two cycles that need to be understood:

- The purge cycle, refers to a period which may involve multiple executions of the purge scripts.
- The maintenance cycle, refers to the number of days required to perform all space management operations (refer B.1).

The goal is to execute the purge scripts and maintenance operations to determine the optimal cycle for each ensuring that the purge has headroom to grow. The purge script may need to be executed multiple times per day and maintenance operations may span multiple days.

These sections will help to tune, monitor and reclaim space:

- Parallel purge (refer 3.1)
- Monitor the hardware and database (refer A.2)
- Space management operations (refer B.1)

2.2.5. Review the Testing results.

The goals should be:

- Determine if the purge is able to delete and reclaim space, either equal to or above the inflow (refer 1.2 & 1.3).
- Determine how many times the purge must be executed and the total elapse time of the purge cycle to meet purging requirements. This purge cycle needs to complete within 80% of the time allocate for its execution.
- Determine the bottleneck tables (refer 3.1.4).
- Review database and O/S reports for resource contention and waits (refer A.2).

2.2.6. Partition the tables which are a bottleneck.

Large installations should consider partitioning the tables which have shown to be difficult to purge, these tables might also be the largest tables. Table partitioning is a proven method of removing bulk data and the SOA schema is instrumented with partition key to facilitate range partitioning.

As of SOA PS5 the granularity of tables that can be partitioned has improved. It is now possible to partition high volume tables with only their master table, this should make partitioning more attractive and feasible:

- Partial partitioning (refer C.3.3.2)
- Long running composites (refer 1.4)

2.2.6.1. Base point backup with partitioned table.

2.2.7. Repeat the purge testing and review (2.2.4/5) excluding the partitioned table.

2.3. Recommended Strategies.

The growth management strategies recommended in this section is based on the SOA database profile.

The strategies require testing in order to determine their effectiveness. The testing needs to ensure that the purge scripts have headroom grow; that partitioned tables have sufficient disk space; that growth is monitored and that maintenance is performed with minimal impact to OLTP performance.

2.3.1. Recommendations for Large Database profiles

Installation with large database profile must seriously consider table partitioning as part of their strategy. Table range partitioning is a proven method for managing large tables:

1. Execute the parallel purge excluding the tables that will be partitioned (refer C.2)
2. Partition the bottleneck tables to the purge script (refer C.3).
3. Reclaim space as part of maintenance cycle (refer B.1).
4. Drop eligible partitions as part of the maintenance cycle (C.3.2).
5. Monitor the inflow and outflow (refer 1.2 & 1.3).
6. Monitor database and hardware resources (refer B.2).

2.3.2. Recommendations for Medium Database profiles

The installations with a medium database profiles have the potential to become large thus the recommendations are the same as that for large profiles (refer 2.3.1).

It is important that medium installation perform rigorous QA testing and monitoring, as table partitioning post go-live might require downtime.

The only additional recommendation is that the CTAS scripts when available may be an alternative to purging. This depends on the size of the database being truly medium and the availability of downtime during the maintenance windows.

2.3.3. Recommendations for Small Database profiles

Small installation with limited CPU, memory and disk space should start with the single threaded purge and then move to the parallel purge:

1. Execute the single threaded purge (refer C.1)
2. Execute the parallel threaded purge (refer C.2)
    a. If the single threaded purge is not performing and then the parallel purge should be tested.
3. Monitor the inflow and outflow (refer 1.2 & 1.3).
4. Monitor database and hardware resources (refer B.2).

The only additional recommendation is that the CTAS scripts when available may be an alternative to purging. This depends on the size of the database being truly small and the availability of downtime during the maintenance windows.

3. Troubleshooting.

- Parallel Purge
    - Degree of Parallel
    - Parsing composite to delete.
    - Parallel Query Slaves.
    - Debug and Tracing.
- SOA Table Partitioning
    - Referential Integrity and Equi-Partitioning
    - Composite Range-Hash partitions.
    - Interval Partitioning.
    - Global Hash Indexes.
    - Partition Pruning.
    - Purging Partitions.
- Auditing

3.1. Parallel Purge

The recommendations in this section are primarily to help address escalations where the SOA database has not been managed and space need to be reclaimed. The advice should also assist with determining the appropriate parameter settings for normal daily execution.

When trying to reclaim space the purge will need to be executed frequently, in an effort to delete substantially more composites than are being created. This has implications for host and database resources which need to be monitored to avoid length wait queues (refer B.2).

There should be a clear goal on the number of composites to delete otherwise the purge can be configured too aggressively. The areas that will be highlighted:

- Degree of Parallel.
- Parsing Composites to Delete.
- Parallel Query Slaves.
- Debug and Tracing.

Note: Metalink Support NOTE 358.1 should be reviewed for purge script patches.

3.1.1. Degree of Parallel

When selecting the appropriate Degree of Parallel (DOP) hardware resources need to be considered. The DOP will determine the number of jobs that are scheduled by the database and thus the number of processes concurrently executed on the host.

The parallel purge jobs are I/O intensive which means that they are not entirely CPU bound; however the following formula is a sensible starting point after which test and monitoring will be required to determine any increase:

DOP <= PARALLEL_THREADS_PER_CPU x CPU_COUNT

While the purge jobs are run concurrently they are actually executed with a 30 second incremental delay to try and avoid resource contention (The delay is not meant to avoid row contention as the purge jobs should not contend for the same composites). This delay has implications for the elapse time of the purge jobs, as an example:

- A DOP of 10 means that the 10th purge job will sleep for 270 seconds before starting (i.e.; (DOP - 1) * 30 sec).

If a high DOP numbers is specified then the “max_runtime” parameter must be set appropriately.

3.1.2. Parsing Composites to Delete

The temptation to delete as many composites as possible in each execution of the purge script is understandable when trying to reclaim space. However the script must parse the data for appropriate composite which can monopolize the elapse time of the purge.

The strategy should be to chip away at the data by executing the purge often with modest date and max_count parameter settings. This chipping away requires the parallel purge to be scheduled very frequently, maybe even twice an hour.

The set of composites to parse is determined by parameters:

- min_creation_date,
- max_creation_date,
- retention_date
- max_count

A modest max_count parameter setting is the most effective method to reduce parse elapse times. The max_count should be set low, maybe 50k to start with until an optimal value can be determined. As the tables start to reduce in size, the max_count and date ranges can increase.

Unfortunately this method of chipping away has not always proven effective and on occasions there has been a need to introduce Parallel Query in order to parse larger amounts of data.

3.1.3. Parallel Query Slaves

The parsing for candidate composites to delete is primarily performed by a few "INSERT ... SELECT" statements. The larger the set of composites to parse the more the database Cost Based Optimizer

(CBO) will favor Full Table Scan (FTS). When large portions of a table need to be traversed, a FTS can be faster than index access as it performs multi-block I/O calls.

Parallel Query (PQ) can significantly improve the performance of FTS by coordinating and distributing the work over multiple slave processes. Additionally, the PARALLEL hint on the INSERT portion will execute in APPEND mode, further improving performance through Direct Path Inserts.

As with the scheduled purge jobs, the PQ Slaves are I/O intensive so are not entirely CPU bound; however the following formula is a good starting point when selecting a DOP for the PARALLEL hint:

DOP <= PARALLEL_THREADS_PER_CPU x CPU_COUNT

Note: There is a DOP setting to determine the number of parallel purge jobs and a DOP to determine the number of Parallel Query slaves. These DOP settings will both impact the amount of processes executed on the host but not simultaneously. The "INSERT ... SELECT" statement will execute first to parse the data followed by the scheduling of the parallel purge jobs.

For the "INSERT ... SELECT" statement, the PARALLEL hint can be specified after the INSERT keyword and/or after the SELECT keyword. Thus, parallelism of the INSERT and SELECT operations are independent of each other. The appropriate DOP for the entire statement will be determined by the following precedence rule:

- PARALLEL hint on INSERT portion of the statement.
- Set at Session.
- DOP on INSERT Table.
- Maximum DOP set on any table in the statement..

In order to execute Parallel DML the following statement will need to be executed within the script:

“ALTER SESSION FORCE PARALLEL DML statement”

Example:

The statement has A DOP of 2 for the INSERT and 4 for the SELECT, however given the precedence rules the DOP for the entire statements will be 2:

```
INSERT /*+ PARALLEL(tbl_ins,2) */ INTO tbl_ins
SELECT /*+ PARALLEL(tbl_sel,4) */ * FROM tbl_sel;
```

Note: Parallel Query can also be applied to the DELETE and SUBSELECT in the purge script, if required.

3.1.4. Debug and Tracing

Debug:

The purge scripts are instrumented with a DEBUG flag that provides detailed time stamped information on each operation. This information can be evaluated to determine which tables are difficult to delete and may be better managed outside of the purge via table partitioning.

To set the DEBUG flag:

Make an O/S directory where the debug logs will be written:

```
Host> mkdir -p /…/debuglogs
```

Connect to the database as SYSDBA privileged account to create the database directory

```
SQL> create or replace directory SOA_PURGE_DIR as '/.../debuglogs';
```

Grant privilege to the SOA schema owner:

```
SQL> grant read, write on directory SOA_PURGE_DIR to dev_soainfra;
```

Connect as SOA Schema Owner and set the DEBUG flag.

(MW_HOME/SOA_ORACLE_HOME/rcu/integration/soainfra/sql/soa_purge/common)

```
SQL> @debug_on.sql;
```

Execute the purge script.

```
SQL> set severout on
SQL> <execute purge>
```

Note: As an alternative the SQLPLUS spool command can be used to capture the debug information.

The AWR reports for the period of the purge can also help to determine the SQL with the longest Elapse and Execution times.

Trace:

If SQL tracing is required to diagnose the performance of the parallel purge scripts then the following will be required:

Edit procedures "soa.delete_instances_in_parallel" & "soa.delete_insts_in_parallel_job" to add the SQL trace (10046):

```
execute immediate 'ALTER SESSION SET EVENTS ''10046 trace name context forever, level 12''';
```

Note: All quotes are single quotes.

This will create a SQL trace for the purge session and the scheduled purge jobs (J000) in the database user dump destination.

```
SQL> show parameter dump
```

The trace for the main database session can be identified by searching the trace files for string "dbms_scheduler.create_job" or "delete_instances_in_parallel":

```
Host> grep -i dbms_scheduler.create_job *
```

- E.g.: TESTORA2_ora_3893.trc

The number of scheduled jobs will depend on the Degree of Parallel:

E.g.: A DOP of 4:

- TESTORA2_j000_9585.trc
- TESTORA2_j001_9587.trc
- TESTORA2_j002_9604.trc
- TESTORA2_j003_9606.trc

If Parallel Query slaves have been customized into the purge script then there will be a trace files (P000) for each. These PQ trace files are not as important, as the query plans in the other trace file will account for their execution

- E.g.: PSOAORA2_p000_4284.trc,...

TKPROF the trace files and sort by elapse time to determine the most costly SQL:

```
Host> tkprof TESTORA2_ora_3893.trc ../out/ TESTORA2_ora_3893.out
waits=yes sys=yes aggregate=yes sort=exeela
```

3.2. SOA Table Partitioning.

Oracle table partitioning addresses many concerns about the maintenance of large tables by decomposing them into smaller, more manageable segments called partitions. The SOA Schema has been instrumented with partition keys to take advantage of the Oracle database RANGE partitioning.

3.2.1. Referential Integrity and Equi-Partitioning

For performance reasons the SOA components (excluding B2B) have no foreign key (FK) constraints to police referential integrity. Without these constraints the relationship between master and detail tables must be protected to avoid dangling references in the detail tables.

The term equi-partitioning, is a database partitioning feature that partitions master and detailed tables based on their FK constraint. This feature creates table partitions that group related master and detail rows into the same date range. This grouping ensures no dangling references are created when partitions are dropped.

To mimic this feature the SOA engine has been instrumented to push the creation date of the composite instances (Fabric) down into the component masters tables (BPEL, Mediator, HWF). As Fabric is the top level component on which all other components depend, this method will group all dependent rows by date range.

As an example; the timestamp in the “created_time’ column of the “composite_instance” table is pushed down into:

- BPEL master table “cube_instance” column “cpst_inst_created_time” which is then pushed down into dependent tables like “cube_scope.ci_partiton_date”.
- Mediator master table “mediator_instance” column “composite_creation_date” which is then pushed down into dependent tables like: “mediator_case_instance.mi_partition_date”

Note:

- Dependent tables must be partitioned with their master and in all cases the top level Fabric “composite_instance” table must be partitioned.
- To complete the equi-paritioing the table partitions must all share the same name and date range.

In summary, equi-partitioning means that the associated dependent table rows are in a partition with the same partition key range as their master table rows. Thus the state of each detail row in the equi-partition can be inferred from its associated master table row.

3.2.2. Composite Range-Hash partitions

Range partitions can have their rows hashed into a sub-partition to implement composite range-hash partitioning. Hash sub-partitions can be of benefit to distributing I/O but it is not currently recommended when partitioning the SOA tables.

Hashing of keys is a method which works well with equality predicates ("=", "IN") and there are a few tables with keys that are good candidates like; the BPEL CIKEY. However range-hash partitions alone do not convey the uniqueness of the hash key. For the CBO to determine uniqueness the query would need to search all partitions of a table (Full table Scan (FTS)). Partition pruning would help by narrowing the range of partitions to search but SOA does not take advantage of this feature (refer 3.2.5). Thus to avoid a FTS when the CBO is determines uniqueness, indexes are searched bypassing the hash sub-partition.

3.2.3. Interval Partitioning.

Interval partitioning is an extension of range partitioning which instructs the database to automatically create partitions as required.

Presently the verification scripts (refer 3.3.) require the partition name to be the same across all table in a group. Interval partitioning system generates partition names which will differ for each partition, so is not current supported.

3.2.4. Global Hash Indexes.

The use of Global Hash Indexes is independent of the SOA table partitioning.

The SOA engine is constantly inserting rows and many of the index keys are monotonically increasing. Indexes of a BTREE structure will insert these keys targeting only a few database blocks which can become very hot across a Real Application Cluster (RAC). This is usually seen in the AWR reports as excessive "Buffer Busy Waits". To distribute the index keys randomly across many database blocks Global Hash Indexes can be used. (Please refer "Oracle FMW 11g R1 SOA with Oracle Database Real Application Clusters Assessment" in the reference section).

3.2.5. Partition Pruning

When the range partition keys are used as predicates, the optimizer can prune the number of partitions to search. The SOA engine and especially the EM Console do not presently use partition keys in SQL queries and thus does not take advantage of this performance feature.

With the requirements of equi-partitioning (refer 3.2.1) the partition keys are populated with the composite creation date which is not the creation date of the components. The components (BPEL, Mediator, HWF) have their own creation dates which are used in the console queries:

Example for BPEL:

- Partition Key: Table "cube_instance" column "cpst_inst_created_time"
- Creation Date: Table "cube_instance" column "creation_date"

3.2.5. Purging Partitions

Circumstances may arise that require the partitioned tables to be purged, especially if the row migration scripts are used. This defeats the purpose of partitioning the table for maintenance which can remove data in bulk via the "alter table … drop partition" command.

Attempting to purge the partitioned tables will most likely suffer the parsing performance problem as described in section 3.1.2. If partitions need to be purged then shrinking the partitions should also be performed, refer appendix B.

3.3. Reducing Audit levels

This section provides tuning information on how to reduce the audit level for composites and thus the data written to the SOA schema.

All production environments should have the Audit level set to Production. This should be standard setting for all SOA domains in production. Please note that each component can be set independently of the composite level however Production should be the minimal of settings. Various performance tests have shown that changing from the audit level from *Development* to *Production* results in an average of 34 % - 46 % performance improvement (without tuning the individual component audit settings).

If an organization can turn off auditing (provided the business allows for it) there are substantial performance improvements. Internal performance results have shown the changing from *Production* to *Off* can result in an average improvement of over 50%-60% (without tuning the individual component audit settings). However since most organizations cannot turn off auditing we recommend the following steps to be considered for reducing the audit levels.

For synchronous BPEL components if the audit trail is not required then use the following settings:

- inmemoryOptimization = true
- completionPersistPolicy = faulted

With these settings the synchronous process only shows in the audit trail if the instance faults. Please note these settings should only be used for synchronous BPEL processes and not for any type of asynchronous BPEL processes.

Another way to reduce the audit level is to turn instance state tracking. On the FMW console Uncheck the "Capture Composite Instance State" this has two benefits

1. A performance improvement since the state is not captured in the DB
2. Less data in the DB for purge purposes.

The implication is although you may still see the composite instances in the FMW console, the "state" of composite instances will be shown as a question mark (?), so you cannot decipher from the console whether the composite has completed or not. Please note that the state column will not reflect the states of the individual components (BPM, BPEL, UserTask, Mediator, etc) within the composite. The state of the individual components can be tracked by going to the respective component pages for e.g. to soa-infra → Service Engine → BPEL in FMW console.

Each component also has its own audit level which can be set since the initial setting is Inherit, which will inherit the global audit setting. These settings can be set via FMW console soa-infra →SOA Administration → BPMN (BPEL, Mediator) Properties. Auditing can be further tuned by setting the level of the composite audit level. It is also set to Inherit by default and it inherits the respective component auditing level. So the following settings are being used at several customers:

- Set global audit level = Production
- Capture Composite Instance State = Off
- BPEL Engine Audit Level = Minimal
- Composite Audit Level = Inherit
- InMemoryOptimization/completionPersistPolicy = true/faulted (for synchronous processes only!)

The above settings have reduced the composite data traffic by 50% at many customer sites and drastically improved purging times.

Conclusion

Capacity planning and testing with consideration for future growth is paramount to a well designed SOA database growth management strategy. Decision should not be limited by current requirements alone which may fall short in production and require downtime to resolve. The need for database maintenance and OLTP performance must be balanced, as both are equally important.

References

- Oracle® Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite 11g Release 1
- Oracle® Database VLDB and Partitioning Guide 11g Release 2 (11.2)
- Oracle® Database Performance Tuning Guide 11g Release 2 (11.2)
- Oracle® Database Administrator's Guide 11g Release 2 (11.2)
- Oracle® Enterprise Manager Administration 10g Release 5 (10.2.0.5)
- Oracle FMW 11g R1 SOA with Oracle Database Real Application Clusters Assessment http://www.oracle.com/technetwork/database/focus-areas/availability/maa-fmw-soa-racanalysis-427647.pdf
- Oracle BPEL 10g Purging Strategies http://www.oracle.com/technetwork/middleware/bpel/learnmore/bpeldehydrationstorepurgestrategies-192217.pdf
- Oracle BPEL Process Manager 10g Database Schema Partitioning http://www.oracle.com/technetwork/middleware/bpel/documentation/bpel10gpartitioning-133743.pdf
- Maintenance and Administration Advisor: Oracle Fusion Middleware (FMW) SOA 11g Infrastructure Database (Doc ID 358.1)
- OS Watcher Support Note: 301137.1

Appendix A. Monitoring the SOA Installation

Monitor space usage

- Determine the growth trend of the components
- Determine the growth trend of the SOA Schema.
- Determine the largest segments.
- Determine the growth trend of tables and indexes.
- Estimate table growth.
- Estimate index growth.
- Monitor unused indexes

Monitor Hardware/Database:

- Hardware - OS Watcher
- Database -AWR/ADDM
- Disk Storage - Orion.

A.1. Monitor Space Usage

The need to monitor space is an essential task of database and system administrators to avoid unplanned outages. This section provides advice that should be used to help with capacity planning and in determining the effectiveness of the growth management strategy. It is not meant to be a comprehensive guide on all aspects monitoring and capacity planning.

(Please refer to the Enterprise Manager and Database Administration Guide and Database Administration Guide; for more information on capacity planning and monitoring threshold.)

A.1.1. Determine the growth trend of the components.

The component master tables record the creation date of each row which can be used to determine growth trends over a given period:

• composite_instance	created_time
• cube_instance	creation_date
• mediator_instance	created_time

However querying the component master tables is limited as purging will constantly remove data making trend predictions difficult. To determine a trend over a given period requires data to be regularly collected with an associated time stamped. This can be achieved with a simple component history table which is populated each day. This table should be sufficient to determine the effectiveness of the growth management strategy and growth trends.

- For performance reasons it might be best to create a history table per component:

SQL> create table soa_fabric_hist (sfh_date timestamp, sfh_count number);

SQL> insert into soa_fabric_hist select systimestamp, count(*) from composite_instance group by systimestamp;

SQL> create table soa_bpel_hist (sbh_date timestamp, sbh_count number);

SQL> insert into soa_bpel_hist select systimestamp, count(*) from cube_instance group by systimestamp;

Etc.

- An alternate to counting the rows in each component table, is to query the NUM_ROWS column from USER_TABLES as long as the tables are analyzed.

A.1.2. Determine the growth trend of the SOA Schema

The objects (table, index, lobs) created for the SOA schema may be spread over many tablespaces but all have the same owner (*soainfra) which can be used to group space usage. When trying to determine the space requirements of each composite, measuring the whole schema allows for better space metrics as indexes and lob segments are taken into account.

To monitor the growth of the SOA schema a simple history table can be populated daily. This table should be sufficient to determine the effectiveness of space management (refer to Appendix B).

- Create schema growth table based on object type:

  SQL> create table soa_size_schema_hist (ssch_date timestamp, ssch_sum_mb number);

  SQL> insert into soa_size_schema_hist select systimestamp, sum(bytes)/1024/1024 from dba_segments where owner='<soa_owner>' group by systimestamp;

- Optionally, a history table by object type (table,index.lob) may be helpful when coupled with the largest segment to verify which tables may need partitioning or are lacking space management.

  SQL> create table soa_size_type_hist (ssch_date timestamp, ssch_type varchar2(18), ssch_sum_mb number);

  SQL> insert into soa_size_type_hist select systimestamp, segment_type, sum(bytes)/1024/1024 from dba_segments where owner='<soa_owner>' group by systimestamp, segment_type;

.

A.1.3. Determine the largest segments.

Understanding which segments are the largest in the schema will help determine the tables that may be better managed as range partitioned tables. Further, monitoring the largest segments might uncover missing space management operations or inappropriate audit setting.

- To identify the largest 20 segments for the SOA schema.

```
SQL> select segment_name, segment_type, (sum(bytes)/1024/1024) MB_size
from dba_segments where owner='<soa_owner>' and rownum < 20
group by segment_name, segment_type
order by 3 desc;
```

- The largest segments might be LOB segment..

```
SQL> select l.table_name, s.segment_name, (sum(bytes)/1024/1024) MB_size
from dba_lobs l, dba_segments s
where s.owner='<soa_owner>' and s.segment_type = 'LOBSEGMENT'
and s.segment_name = l.segment_name
and s.owner='<soa_owner>"
group by l.table_name, s.segment_name;
```

OR

```
SQL> select l.table_name from dba_lobs l
where l.owner =’<soa_owner>’ and l.segment_name = ‘<segment_name>’;
```

- To determine the size of an individual segment.

```
SQL> select segment_name, (sum(bytes)/1024/1024) MB_size
from dba_segments where owner='<soa_owner>' and segment_name = ‘<table_name>’
group by segment_name;
```

A.1.4. Determine the growth trend of tables and indexes.

The OBJECT_GROWTH_TREND function of the DBMS_SPACE package can be used to show the trend in space growth for a table (Refer to the Database Administration guide for more information.).

The OBJECT_GROWTH_TREND function of the DBMS_SPACE package produces a table of one or more rows, where each row describes the space use of the object at a specific time. The function retrieves the space totals from the Automatic Workload Repository (AWR) or computes current space and combines it with historic space changes retrieved from Automatic Workload Repository.

The following example displays the growth in used and allocated space over time for the COMPOSITE_INSTANCE table.

```
SQL> select timepoint, space_usage, space_alloc, quality
        from table (dbms_space.object_growth_trend
        ('DEV_SOAINFRA','COMPOSITE_INSTANCE','TABLE'));
```

Complete Syntax for DBMS_SPACE.OBJECT_GROWTH_TREND procedure

```
dbms_space.object_growth_trend (
object_owner in varchar2,
object_name in varchar2,
object_type in varchar2,
partition_name in varchar2 default null,
start_time in timestamp default null,
end_time in timestamp default null,
interval in dsinterval_unconstrained default null,
skip_interpolated in varchar2 default 'false',
timeout_seconds in number default null,
single_datapoint_flag in varchar2 default 'true')
```

Description of OBJECT_GROWTH_TREND Function Parameters

TABLE 2 OBJECT_GROWTH_TREND FUNCTION PARAMETERS

PARAMETER	DESCRIPTION
OBJECT_OWNER	The schema containing the object
OBJECT_NAME	The name of the object
OBJECT_TYPE	The type of the object
PARTITION_NAME	The name of the table or index partition, is relevant. Specify NULL otherwise
START_TIME	A TIMESTAMP value indicating the beginning of the growth trend analysis
END_TIME	A TIMESTAMP value indicating the beginning of the growth trend analysis. The default is "NOW".
INTERVAL	interval ('YES') or not ('NO'). This setting is useful when the result table will be displayed as a table rather than a chart, because you can see more clearly how the actual recording interval relates to the requested reporting interval. The function returns a table, each of row of which provides space use information on the object for one interval. If the return table is very large, the results are pipelined so that another application can consume the information as it is being produced
SKIP_INTERPOLATED	whether interpolation of missing values should be skipped
TIMEOUT_SECONDS	The time-out value for the function in seconds
SINGLE_DATA_POINT_FLAG	whether in the absence of statistics the segment should be sampled

Return Values

```
TYPE object_growth_trend_row IS RECORD(
timepoint timestamp,
space_usage number,
space_alloc number,
quality varchar(20));
```

TABLE 3 OBJECT_GROWTH_TREND_ROW RETURN VALUES

PARAMETER	DESCRIPTION
TIMEPOINT	A TIMESTAMP value indicating the time of the reporting interval. Records are not produced for values of TIME that precede the oldest recorded statistics for the object.
SPACE_USAGE	The number of bytes actually being used by the object data
SPACE_ALLOC	The number of bytes allocated to the object in the tablespace at that time.
QUALITY	A value indicating how well the requested reporting interval matches the actual recording of statistics. This information is useful because there is no guaranteed reporting interval for object size use statistics, and the actual reporting interval varies over time and from object to object • GOOD: The value whenever the value of TIME is based on recorded statistics with a recorded timestamp within 10% of the INTERVAL specified in the input parameters. (The output returned by this function is an aggregation of values recorded across all instances in an Oracle RAC environment. Each value can be computed from a combination of GOOD and INTERPOLATED values. The aggregate value returned is marked GOOD if at least 80% of that value was derived from GOOD instance values.) • INTERPOLATED: The value did not meet the criteria for GOOD, but was based on recorded statistics before and after the value of TIME. Current in-memory statistics can be collected across all instances in a cluster and treated as the "recorded" value for the present time • PROJECTION: The value of TIME is in the future as of the time the table was produced. In an Oracle Real Application Clusters environment, the rules for recording statistics allow each instance to choose independently which objects will be selected.

A.1.5. Estimate Table size.

The CREATE_TABLE_COST procedure of the DBMS_SPACE package allows the size of a table to be estimated using the predicted number of rows for an existing table or based on a table column definition. The size of tables can vary widely based on the table space storage attribute which is accounted for by this procedure. There are two overloads of this procedure:

- The first procedure takes the column information of the table.
- The second procedure takes the average row size of the table.

As the SOA tables already exist, it is the second variant that will be most useful. The following example will estimate the size of the CUBE_INSTANCE table with 10.000 rows; an average row length of 360 and PCT FREE 0f 10.

- The average row length and current PCT_FREE can be derived from DBA_TABLES after ensuring that the table has been analyzed.

  ```
  SQL> select avg_row_len, pct_free from dba_tables where table_name =
  'CUBE_INSTANCE' and owner = 'DEV_SOAINFRA';
  ```

- Execute the DBMS_SPACE.CREATE_TABLE_COST procedure.

  ```
  set serverout on
  declare

     v_used number(10);
     v_alloc number(10);

  begin

     DBMS_SPACE.CREATE_TABLE_COST
        ('SOA1_SOAINFRA',360,10000,10, v_used,v_Alloc);
     dbms_output.put_line('used bytes: ' || to_char(v_used));
     dbms_output.put_line('allocated bytes: ' || to_char(v_alloc));

  end;
  /
  ```

The following example estimates the space usage of a table based on its column definition.

```
set serveroutput on
declare

   v_used_bytes number(10);
   v_allocated_bytes number(10);
   v_type sys.create_table_cost_columns;

begin

   v_type := sys.create_table_cost_columns
   ( sys.create_table_cost_colinfo('number',9),
    sys.create_table_cost_colinfo('varchar2',50),
    sys.create_table_cost_colinfo('varchar2',15),
    sys.create_table_cost_colinfo('date',null),
    sys.create_table_cost_colinfo('date',null) );

   dbms_space.create_table_cost
         ('users',v_type,10000,7,v_used_bytes,v_allocated_bytes);
   dbms_output.put_line('used bytes: ' || to_char(v_used_bytes));
   dbms_output.put_line('allocated bytes: ' || to_char(v_allocated_bytes));
```

```
end;
/
```

Complete Syntax DBMS_SPACE.CREATE_TABLE_COST procedure

First Overload:

```
dbms_space.create_table_cost (tablespace_name in varchar2,
   colinfos in create_table_cost_columns, row_count in number,
   pct_free in number, used_bytes out number,
   alloc_bytes out number);
   create type create_table_cost_colinfo is object (col_type varchar(200),
   col_size number);
```

Second Overload:

```
dbms_space.create_table_cost (tablespace_name in varchar2,
   avg_row_size in number, row_count in number, pct_free in number,
   used_bytes out number, alloc_bytes out number);
```

Both variants require as input the following values:

TABLE 4 CREATE_TABLE_COST PROCEDURE PARAMTERS

PARAMETER	DESCRIPTION
TABLESPACE_NAME	The tablespace in which the object will be created. The default is the SYSTEM tablespace.
ROW_COUNT	The anticipated number of rows in the table.
PCT_FREE	The percentage of free space you want to reserve in each block for future expansion of existing rows due to updates. In addition, the first variant also requires as input a value for AVG_ROW_SIZE, which is the anticipated average row size in bytes. The second variant also requires for each anticipated column values for COLINFOS, which is an object type comprising the attributes COL_TYPE (the data type of the column) and COL_SIZE (the number of characters or bytes in the column).

TABLE 5 CREATE_TABLE_COST PROCEDURE RETURN VALUES

PARAMETER	DESCRIPTION
USED_BYTES	The actual bytes used by the data, including overhead for block metadata, PCT_FREE space, and so forth.

ALLOC_BYTES	The amount of space anticipated to be allocated for the object taking into account the tablespace extent characteristics

A.1.6. Estimating Index size.

The CREATE_INDEX_COST procedure of the DBMS_SPACE package lets you estimate the space use cost of creating an index on an existing table. This can be used to determine the cost of adding an index to the SOA schema.

The following example will estimate the size of a new index based on DDL.

```
set serveroutput on
declare

  v_used_bytes number(10);
  v_allocated_bytes number(10);

begin

  dbms_space.create_index_cost(
    'create index cube_index on cube_instance(cikey)'
    v_used_bytes, v_allocated_bytes);
  dbms_output.put_line('used bytes: ' || to_char(v_used_bytes));
  dbms_output.put_line('allocated bytes: '|| to_char(v_allocated_bytes));

end;
/
```

Complete Syntax DBMS_SPACE.CREATE_INDEX_COST procedure

```
dbms_space.create_index_cost (ddl in varchar2, used_bytes out number,
   alloc_bytes out number, plan_table in varchar2 default null);
```

The procedure requires as input the following values:

- DDL: The CREATE INDEX statement that would create the index.
- DDL statement must be against an existing table.
- [Optional] PLAN_TABLE: The name of the plan table to use. The default is NULL. The results returned by this procedure depend on statistics gathered on the segment. Therefore, be sure to obtain statistics shortly before executing this procedure. In the absence of recent statistics, the procedure does not issue an error, but it may return in appropriate results. The procedure returns the following values:
- USED_BYTES: The number of bytes representing the actual index data.
- ALLOC_BYTES: The amount of space allocated for the index in the table space.

Usage Notes

- The table on which the index is created must already exist.
- The computation of the index size depends on statistics gathered on the segment.
- It is imperative that the table must have been analyzed recently.
- In the absence of correct statistics, the results may be inaccurate, although the

A.1.7. Monitor unused Indexes.

Oracle Database provides a means of monitoring indexes to determine whether they are being used. If an index is not being used, then it can be dropped, eliminating unnecessary statement overhead.

To start monitoring the usage of an index, issue this statement:

SQL> alter index *index_name* monitoring usage;

Later, issue the following statement to stop the monitoring:

SQL> alter index *index_name* nomonitoring usage;

The view V$OBJECT_USAGE can be queried for the index being monitored to see if the index has been used. The view contains a USED column whose value is YES or NO, depending upon if the index has been used within the time period being monitored. The view also contains the start and stop times of the monitoring period, and a MONITORING column (YES/NO) to indicate if usage monitoring is currently active. Each time that you specify MONITORING USAGE, the V$OBJECT_USAGE view is reset for the specified index. The previous usage information is cleared or reset, and a new start time is recorded. When you specify NOMONITORING USAGE, no further monitoring is performed, and the end time is recorded for the monitoring period. Until the next ALTER INDEX...MONITORING USAGE statement is issued, the view information is left unchanged.

A.2. Monitor Hardware and Database.

The hardware resource must be able to support the tools selected to implement the growth strategy. It is important to monitor hardware resource during OLTP and maintenance periods and especially when these periods overlap.

A.2.1. Hardware - OS Watcher

To monitor CPU, Disk, Memory and Network O/S resources Oracle provides OS Watcher (OWS) which should be installed on all tiers in the SOA installation.

OWS is a collection of UNIX shell scripts intended to collect and archive operating system and network metrics to aid support in diagnosing performance issues. OSW operates as a set of background processes on the server and gathers OS data on a regular basis, invoking such Unix utilities as vmstat, netstat and iostat.

The OSW user guide and product can be downloaded via Metalink Support note 301137.1.

A.2.2. Database – AWR / ADDM

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

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 2 Day Performance Tuning Guide for information about using Oracle Enterprise Manager.).

A.2.3. Disk I/O - Orion

A complete SOA I/O design plan is beyond the scope of this paper but good advice can be found in the Oracle Database performance tuning guide. A well designed I/O plan will consider capacity, availability and performance.

The AWR reports will identify segments, data files and tablespaces that are accessed frequently and OS watcher will identify lengthy I/O queues. The combination of these tools will identify the disk devices and files of concern. This might identify a need to redistribute files in order to balance I/O or to modify the RAID levels on slow volumes.

The I/O calibration feature (Orion) of Oracle Database issues random I/O using the Oracle data files to access the storage media, producing results that closely match the actual performance of the database. Orion is expressly designed for simulating Oracle database I/O workloads using the same I/O software stack as Oracle. Orion can also simulate the effect of striping performed by Oracle Automatic Storage Management. (Please refer to the Oracle Database Performance guide for more information)

Appendix B. Space Management

Much of the information in this section can be found in the various Oracle Database Administration Guides. The space management concepts and commands summarized in this section are to address a common misconception that the SOA purge scripts will reclaim space. To reclaim space database maintenance operations need to be executed.

By default the SOA schema is created on locally managed table spaces with automatic segment space management, so all advice will be limited by this. This section is not meant as a comprehensive guide to all database space management features.

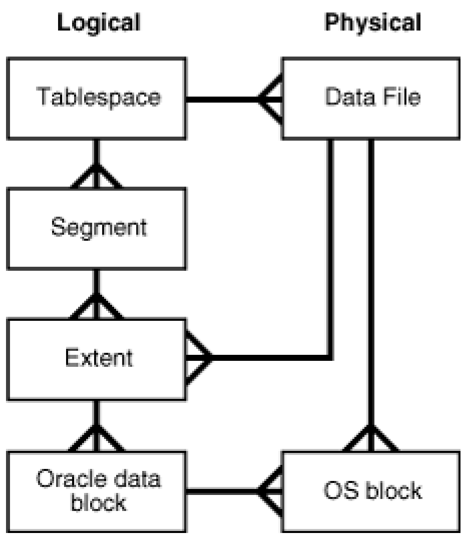
B.1. Concepts

A Data files can be broken down into the following components:

- Segment
  A segment contains a specific type of database object. That is; a table is stored in a table segment, and an index is stored in an index segment..
- Extent
  An extent is a contiguous set of data blocks within a segment. Oracle Database allocates space for segments in units of one extent.
- Data block
  A data block, also called a database block, is the smallest unit of I/O to database storage. An extent consists of several contiguous data blocks.
- Tablespace:
  A tablespace consists of one or more physical data files. A locally managed tablespace maintains a bitmap in the data file header to track free and used space in the data file body. Each bit corresponds to a group of blocks. When space is allocated or freed, Oracle Database changes the bitmap values to reflect the new status of the blocks

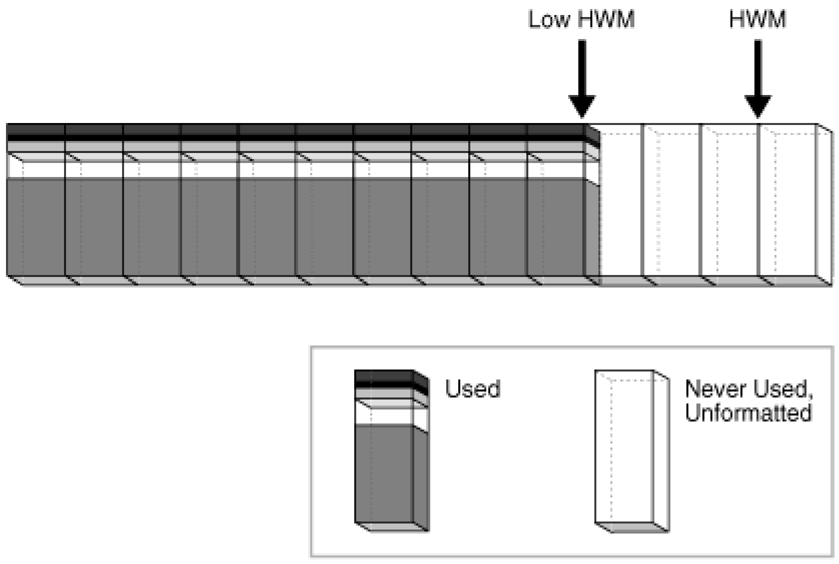
Segments, extents, and data blocks are all logical structures.

Logical and Physical Storage



SEGMENT HIGH WATER MARK

To manage space, Oracle Database tracks the state of blocks in the segment. The high water mark (HWM) is the point in a segment beyond which data blocks are unformatted and have never been used. When the blocks between the HWM and low HWM are full, the HWM advances to the right and the low HWM advances to the location of the old HWM. As the database inserts data over time, the HWM continues to advance to the right, with the low HWM always trailing behind it. Unless you manually rebuild, truncate, shrink/de-allocate the object, the HWM never retreats.



B.2. Reclaiming Segment and Data file Space

The following will provide instructions on the common techniques used to reclaiming Segment and Data file space.

- Segment Space:
  - Online segment shrink.
  - De-allocate unused space.
  - Index rebuild and coalesces
  - Table partitions
  - Secure file LOBS
    - Requirements
    - Conversion
    - Migration
  - Other techniques (CTAS and truncate)
- Data file resize

B.2.1. Online Segment Shrink

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. The space can be defragmented and extents reclaimed by performing an online segment shrink. 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.

DML can still be issued during the data movement phase of segment shrink. However DML operations are blocked for a short time at the end of the shrink operation when the space is de-allocated. Indexes are maintained during the shrink operation and remain usable.

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)

General Online Segment Shrink Steps

- To shrink a segment

  SQL> alter table *table_name* shrink space;

- Before executing the shrink command row movement need to be enables.

  SQL> alter table *table_name* enable row movement;

- The COMPACT clause lets you divide the shrink segment operation into two phases. When you specify COMPACT, the segment space is defragmented and rows are compacted but it postpones the resetting of the high water mark and the de-allocation of the space. Dividing the operations into two should be useful for large tables and will reduce the impact on the blocking of DML during the de-allocation phase.

  You can reissue the shrink space without the COMPACT clause during off-peak hours to complete the second phase.

  SQL> alter table *table_name* shrink space compact;

- The CASCADE clause extends the segment shrink operation to all dependent segments of the object. For example, if you specify CASCADE when shrinking a table segment, all indexes of the table will also be shrunk.

  SQL> alter table *table_name* shrink space compact cascade;

- For very large tables it would be advisable to perform the shrink in two phases and to not use the CASCADE clause. Perform the COMPACT operation first, maybe even compact the BASIC LOBS before that, then execute the normal shrink command to reclaim the unused space.

- All segment types are eligible for online segment shrink except the following:
  - IOT mapping tables
  - Tables with rowid based materialized views
  - Tables with function-based indexes
  - Securefile LOBs
  - Compressed tables

Examples

- Shrink a large table in two phases:

  SQL> alter table *table_name* enable row movement;

  SQL> alter table *table_name* shrink space compact;

  SQL> alter table *table_name* shrink space;

- Shrink a table and all of its dependent segments (including BASICFILE LOB segments):

  SQL> alter table *table_name* shrink space cascade;

- Shrink a BASICFILE LOB segment only:

  SQL> alter table *table_name* modify lob (*lob_name*) (shrink space);

- Shrink a single partition of a partitioned table:

  SQL> alter table *table_name* modify partition *p1* shrink space;

B.2.2. De-allocate unused space

The DEALLOCATE UNUSED command can be used to manually de-allocate unused space. This command will free unused space above the High Water Mark (The Online segment Shrink also releases space above HWM.).

SQL> alter table *table_name* deallocate;

The optional KEEP clause to specify the amount of space retained in the segment of table, index or cluster:

SQL> alter table *table_name* deallocate unused keep integer;

SQL> alter index *index_name* deallocate unused keep integer;

Note: The UNUSED_SPACE procedure of the DBMS_SPACE package, returns information about the position of the high water mark and the amount of unused space in a segment. For segments in locally managed table spaces with automatic segment space management, use the SPACE_USAGE procedure for more accurate information on unused space. (The DBA_FREE_SPACE view can be used to verify the de-allocated space.)

B.2.3. Index Rebuild and Coalesce.

Whether to rebuild BTREE indexes or not is a point of contention for Database Administrators. Unfortunately over time the SOA purge scripts will fragment most of the SOA BTREE indexes in a manner which requires them to be rebuilt to maintain SQL performance.

The purge scripts delete only closed composites, leaving the open ones in each index data block. As many of the SOA index keys are monotonically increasing the free space in the data block will not be reused.

SQL> alter index *index_name* rebuild or coalesce

TABLE 6 COST AND BENEFITS OF COALESCING OR REBUILDING INDEXES

REBUILD INDEX	COALESCE INDEX
Quickly moves index to another tablespace	Cannot move index to another tablespace.
Higher costs: requires more disk space	Lower costs: does not require more disk space
Creates new tree, shrinks height if applicable	Coalesces leaf blocks within same branch of tree
Enables you to quickly change storage and tablespace parameters without having to drop the original index	Quickly frees up index leaf blocks for use.

On method to combat the need for index rebuild is to convert them to Global Hash Indexes. Hashing monotonically increasing keys will distribute them randomly across data blocks thus improving space reuse. There are other improvements which include a reduction in 'Buffer Busy Waits" for Hot index blocks but not all SOA indexes are good candidates for conversion. (Please refer to Oracle FMW 11g R1 SOA with Oracle Database Real Application Clusters Assessment.).

B.2.4. Table Partitions

Table partition can be dropped to remove table data in bulk and reclaim space. Within SOA the partitions should not be dropped unless deemed eligible, refer the verification scripts C.3.2.

Drop Partition

SQL> alter table table_name drop partition p1;

Though the drop partition will take longer it is worth specifying the UPDATE INDEXES clause to avoid the need to rebuild indexes.

Many table maintenance operations on partitioned tables invalidate (mark UNUSABLE) the corresponding indexes or index partitions. You must then rebuild the entire index or for a global index, each of its partitions. The database lets you override this default behavior if you specify UPDATE INDEXES in your ALTER TABLE statement for the maintenance operation. Specifying this clause tells the database to update the indexes at the time it executes the maintenance operation DDL statement. This provides the following benefits:

The following operations support the UPDATE INDEXES clause:

- add partition , coalesce partition, drop partition, exchange partition, merge partition, move partition, split partition, truncate partition.

Under certain circumstance the following partition operations might be required to assist with customer escalations.

Shrink Table Partition

SQL> alter table *table_name* modify partition *p1* shrink space;

Truncate Table Partitions

SQL> ALTER TABLE ... TRUNCATE PARTITION

Compress table Partition

SQL> alter table *table_name* move partition *part_name* tablespace *tablespace_name* nologging compress for oltp;

Notable restrictions:

- Online segment shrink is not supported for compressed tables.
- Secure File LOBs have their own compression methods (refer
- Compression technology increases CPU resource usage..

- Altering a partition to enable compression applies only to the new data, to compress the existing data you must MOVE the partition. Moving table partitions will drop the old partition segment and creates a new segment, even if you do not specify a new tablespace.

B.2.5. Secure file LOB

Secure files are a new LOB storage architecture which provides performance benefits many factor faster better than traditional LOB access. Secure files are a complete rewrite of the original LOB storage architecture, now called Basic files.

Secure files support advanced features:

- De-duplication - Stores only one copy of identical secure file data.
- Compression - Reduces storage, I/O, redo and encryption overhead. (Note: The Online Segment Shrink command is not supported for Secure files LOBs due to compression.)
- Encryption.

As Secure files are a relatively new feature introduced in Oracle Database 11g, it is recommended that highest available database patch set is applied to avoid know problems.

B.2.5.1. Secure file requirements

The following are required to use secure files.

1. The COMPATIBLE initialization parameter must be set higher than 11.0.0.0.0

```
SQL> show parameter COMPATIBLE;
```

2. The DB_SECUREFILE initialization parameter controls the default action of the database with regards to LOB storage (default if PERMITTED):

```
SQL> alter system set db_securefile = 'always';

SQL> show parameter db_securefile
```

- ALWAYS:
  All LOBs in ASSM table spaces are created as Secure File LOBs. LOBs in non-ASSM table spaces are created as Basic File LOBs unless explicitly specified as Secure Files. Basic File storage options are ignored, and Secure File default storage options are used for any unspecified options.
- FORCE:
  All LOBs are created as Secure File LOBs. If the LOB is being created in a non-ASSM tablespace, an error is thrown. Basic File storage options are ignored, and Secure File default storage options are used for any unspecified options.

- PERMITTED:
  The default setting which allows Secure File LOB storage when the SECUREFILE keyword is used. The default storage method is BASICFILE.
- NEVER:
  Secure File LOBs are not permitted.
- IGNORE:
  Prevent creation of Secure File LOBs, and ignore any errors associated with Secure File storage options.

3. The tablespace is configured to support Automatic Segment Space Management (ASSM).

   SQL> select segment_space_management from dba_tablespaces where tablespace_name = '<*tablespace_name*>';

B.2.5.2. Secure files conversion.

The Repository Creation Utility (RCU) which creates the SOA tables will by default create Basic files, however Secure file LOBs can be created at the time of the SOA schema creations. The process below is only for the time of creations otherwise refer to migration section.

1. Ensure the requirements are met (refer B.2.5.1) and that the db_securefile database initialization parameter is set to ALWAYS or FORCE.

2. Run the appropriate SOA RCU utility to create the schema. Though the LOBs are defined as Basic that will be created as Secure files. The Basic file LOB storage parameters will be ignored.

3. The advanced features of compression, de-duplication and encryption which are not enabled by default:

   To determine if the advanced features are enabled for the secure file LOB column:

   SQL> select table_name, column_name, securefile, retention, encrypt, compression, deduplication from dba_lobs;

   The ALTER command can be used to enable the advanced features should be performed immediately after the RCU:

   SQL> alter table <lob_table> modify lob(<lob_column>) (compress);

   SQL> alter table <lob_table> modify lob(<lob_column>) (deduplicate);

   Note: Oracle recommends that you enable compression, deduplication, or encryption through the CREATE TABLE statement and or Online Redefinition. For existing data, if you enable these features through the ALTER TABLE statement, all Secure Files LOB data in the table is read, modified, and written; this causes the database to lock the table during a potentially lengthy operation. Thus the ALTER table command is not the recommend for converting populated tables and will cause locking.

B.2.5.3. Secure file migration

The Online Redefinition is the recommended online method for converting to Secure File LOBs. However there are offline methods:

- CTAS (Create Table as Select),
- ITAS (Insert Table as Select)
- Export/Import

Advantages of Online Redefinition:

- No requirement to take the table or partition offline
- Can be done in parallel

Disadvantages Online Redefinition:

- Additional storage equal to the entire table or partition and all LOB segments must be available
- Global indexes must be rebuilt

Please refer to the "Migrating Columns from Basic Files LOBs to Secure Files LOBs" section of the Secure Files and Large Objects Developer's Guide for best practice on executing the Online REDFINITON package, Including; preventing redo generation and parallel execution.

B.2.6. Other Techniques

Create table As Select (CTAS)

The CTAS scripts are planned for Oracle SOA PS6. These scripts will only select open composites and thus by default drop all closed composites beyond a given retention period. The process of recreating the table and indexes will also reorganize and reclaim space.

TRUNCATE Statement

The truncate will remove all rows from a table and thus this is unlikely to be performed or required in a SOA production environment.

SQL> truncate table *table_name*;

By default the truncate:

- De-allocates all space used by the removed rows except that specified by the MINEXTENTS storage parameter
- Sets the NEXT storage parameter to the size of the last extent removed from the segment by the truncation process

These truncate options:

- DROP STORAGE, the default option, reduces the number of extents to MINEXTENTS.
- REUSE STORAGE specifies that all space currently allocated is retained.

B.3. Data files resizing.

Firstly to avoid application errors and the need for manual intervention when a tablespace runs out of space, the data files can be set to autoexend. To determine whether a data file is auto-extensible the DBA_DATA_FILES view column AUTOEXTENSIBLE can be queried:

SQL> select autoextensible from dba_data_files

You can specify automatic file extension by specifying an AUTOEXTEND ON clause when the data file is created or altered:

SQL> alter tablespace *tablespace_name* add datafile *'/u01/datafile.dbf'* size 10m autoextend on;

SQL> alter database datafile *'/u01/datafile.dbf'* autoextend on;

RESIZE DATAFILES

If the initial allocation of the data files was excessive or segments were allowed to grow excessively then the data file can be resized. However this is assuming that the space is not being used by segments.

SQL> alter database datafile *'/u01/datafile.dbf''* resize 50m;

As stated it is not always possible to decrease the size of a file to a specific value, in which case the database will return an error (ORA-03297).

Appendix C: Purging / Partitioning Tooling Details

Purge / Partitioning Tooling

This section is not a replacement for reading the SOA Administration Guide. It details some of the SOA PS5 enhancements and row migration procedures.

C.1. Single Thread Purge

The purge scripts will only remove closed composites beyond the retention period. The script is meant for use by small (to medium) profile installations that do not have significant inflow of composites. It has advanced features which allow the setting of a commit batch size and maximum execution times.

Syntax:

```
procedure delete_instances ( min_creation_date in timestamp,
                  max_creation_date in timestamp,
                  batch_size in integer,
                  max_runtime in integer,
                  retention_period in timestamp,
                  purge_partitioned_component in Boolean,
                  composite_name in varchar2,
                  composite_revision in varchar2,
                  soa_partition_name in varchar2 );
```

Note: The following parameters are only available from SOA PS5:

- composite_name,
- composite_revision
- soa_partition_name

TABLE 7 SINGLE THREAD PURGE PARAMETERS

PARAMETER	DESCRIPTION
min_creation_date	Beginning creation date for the composite instances.
max_creation_date	Ending creation date for the composite instances
batch_size	Batch size used to loop the purge. The default value is 20000.

max_runtime	Expiration at which the purge script exits the loop. The default value is 60. This value is specified in minutes.
retention_period	Retention period is only used by the BPEL process service engine (in addition to using the creation time parameter). This functionality is not extended to other components. This parameter checks for and deletes records in the cube_instance table. The value for this parameter must be greater then or equal to max_creation_date. When no value is provided for the retention period, the max_creation_date is considered as the default retention period . Specify a retention period if you want to retain the composite instances based on the modify_date of the BPEL instances (cube_instance table). In this following example, the modify_date of the BPEL instances table, which can be different than the composite created_date, is used as a second level of filtering: • min_creation_date = 1st June 2011 • max_creation_date = 30 June 2011 • retention_period = 1st July 2011 This deletes all composite instances in which the creation_time of the composite is between 1st June 2011 and 30 June 2011 and the modify_date of the cube_instance is less than 1st July 2011
purge_partitioned_component	If set to true the data from tables which have been partitioned will also get purged . The default value is false
composite_name (Available from PS5)	The name of the SOA composite application. This parameter, along with the composite_revision and soa_partition_name parameters, enables you to purge the instances of a specific SOA composite application.
composite_revision (Available from PS5)	The revision number of the SOA composite application.
soa_partition_name(Available from PS5)	The partition in which the SOA composite application is deployed.

Sample Client

```
 DECLARE

   MAX_CREATION_DATE timestamp;
   MIN_CREATION_DATE timestamp;
   batch_size integer;
   max_runtime integer;
   retention_period timestamp;

BEGIN
```

```
  MIN_CREATION_DATE := to_timestamp('2011-01-01','YYYY-MM-DD');
  MAX_CREATION_DATE := to_timestamp('2011-01-31','YYYY-MM-DD');
  max_runtime := 60;
  retention_period := to_timestamp('2011-02-31','YYYY-MM-DD');
  batch_size := 10000;
  soa.delete_instances( min_creation_date => MIN_CREATION_DATE,
                        max_creation_date => MAX_CREATION_DATE,
                        batch_size => batch_size,
                        max_runtime => max_runtime,
                        retention_period => retention_period,
                        purge_partitioned_component => false);
END;
```

SOA PS5 enhancements - Purge by Composite_DN

This feature enables purging of instances of a specific SOA composite which allows certain flows to be purged more frequently. To execute this feature requires the parameters composite_name , composite_revision and partition_name to be supplied.

The following is an example of which targets the 'OrderSDOComposite' composites with a revision of '1.0' and partition name of 'TestSoa'

```
DECLARE
  MIN_CREATION_DATE timestamp := to_timestamp('2011-01-01','YYYY-MM-
DD');
  MAX_CREATION_DATE timestamp := to_timestamp('2011-02-31','YYYY-
MM-DD');
  batch_size integer;
  max_runtime integer;
  retention_period timestamp;
BEGIN
  max_runtime := 60;
  retention_period := to_timestamp('2011-01-31','YYYY-MM-DD');
  batch_size := 10000;
  soa.delete_instances( min_creation_date => MIN_CREATION_DATE,
                        max_creation_date => MAX_CREATION_DATE,
                        batch_size => batch_size,
                        max_runtime => max_runtime,
                        retention_period => retention_period,
                        purge_partitioned_component => true,
                        composite_name => 'OrderSDOComposite',
                              composite_revision => '1.0',
                               soa_partition_name => 'TestSoa');
    END;
```

Considerations Purge by Composite_DN

The purge logic is based on flows (ECIDs) and not COMPOSITE_IDs. Therefore, apart from the intended COMPOSITE_DNs, other composites sharing the same ECID may be deleted. The following two scenarios may occur:

1. Composite Instance is closed but flow still open:
   Taking a scenario where composite A calls composite B and the purge intends to delete instances of composite A. There might be a case where an instance of composite A is closed but the corresponding composite B instance is still open. Since the overall flow is still in an open state the composite A instance will not get purged as the entire flow is not closed.
2. Composite instance is closed, so entire flow is deleted:
   Composite A calls composite B and the purge intends to delete instances of composite A. When composite A and associated Composite B instance are closed the overall flow will be deleted. The intention might have been to delete composite A but both A and B are deleted.

C.2. Parallel Threaded Purge

The parallel threaded purge is functionally the same as the single threaded purge (refer C.1) with one performance advantage that is required for medium to large profile installations. The purge will distribute the workload across multiple jobs to fully utilize host resources and optimize the amount of data that can be deleted in a period.

Syntax:

```
    PROCEDURE delete_instances_in_parallel (
                min_creation_date in timestamp,
                max_creation_date in timestamp,
                batch_size in integer,
                max_runtime in integer,
                retention_period in integer,
                DOP in integer
                max_count integer,
                purge_partitioned_component in Boolean
                composite_name in varchar2,
                composite_revision in varchar2,
                soa_partition_name in varchar2)
```

Note:

- The following parameters are available from PS5:
  - composite_name,
  - composite_revision
  - soa_partition_name
- To execute parallel purge the following privileges have to be granted to the soainfra user :
  - GRANT EXECUTE ON DBMS_LOCK to USER;
  - GRANT CREATE ANY JOB TO USER;

TABLE 8 PARALLEL THREAD PURGE PARAMETERS

PARAMETER	DESCRIPTION
min_creation_date	Beginning creation date for the composite instances.
max_creation_date	Ending creation date for the composite instances
batch_size	Batch size used to loop the purge. The default value is 20000
max_runtime	Expiration time at which the purge script exits the loop. The default value is 60. This value is specified in minutes
Retention_period	Retention period is only used by the BPEL process service engine only (in addition to using the creation time parameter). The default value is null. For more information about this parameter, refer previous section
DOP (Degree of Parallel)	Defines the number of parallel jobs to schedule. The default value is 4.
max_count	Defines the number of rows processed (not the number of rows deleted). A big temp table is created and then jobs are scheduled to purge based on the data. This is the maximum purge row count to use; it defaults to one million. The default value is 1000000.
purge_partitioned_component	If set to true the data from tables which have been partitioned will also get purged . The default value is false
Composite_name (Available from PS5)	The name of the SOA composite application. This parameter, along with the composite_revision and soa_partition_name parameters, enables you to purge the instances of a specific SOA composite application.

Composite_revision(Available from PS5)	The revision number of the SOA composite application.
soa_partition_name (Available from PS5)	The partition in which the SOA composite application is deployed.

Sample Client

```
DECLARE
  max_creation_date timestamp;
  min_creation_date timestamp;
  retention_period timestamp;
BEGIN
  min_creation_date := to_timestamp('2011-01-01','YYYY-MM-DD');
  max_creation_date := to_timestamp('2011-01-31','YYYY-MM-DD');
  retention_period := to_timestamp('2011-02-31','YYYY-MM-DD');
  soa.delete_instances_in_parallel( min_creation_date => min_creation_date,
                                    max_creation_date => max_creation_date,
                                    batch_size => 10000,
                                    max_runtime => 60,
                                    retention_period => retention_period,
                                    DOP => 3,
                                    max_count => 1000000,
                                    purge_partitioned_component => false);
 END;
```

SOA PS5 enhancements - Purge by Composite_DN

Please refer to the section to the Single Threaded purge (C.1) enhancements for details

C.3. SOA Table Partitioning

Oracle table partitioning addresses many concerns about the maintenance of large tables by decomposing them into smaller, more manageable segments called partitions. The SOA Schema has been instrumented with partition keys to take advantage of the Oracle database range partitioning.

C.3.1. Grouping of Tables

The SOA tables can be classified into three groups which are based on functional usage and preserve referential integrity of the data.

Please refer to Table 5 at the end of this appendix for details on the tables within each group.

- **Group 1**

  This group includes tables that are directly related to the end-to-end flow trace of a composite. A majority of the tables fall into this group.

- **Group 1 A**

  This includes a small set of tables that are not directly related to the flow trace.

- **Group 2**

  This includes a small of set of tables that have a dependency on multiple tables from Group 1 and 1A .

C.3.2. Verification Scripts

The verification scripts are required to determine whether a partition can be dropped. The SOA schema maintains referential integrity of partitions via equi-partitioning (refer 3.2.1). These scripts make use of equi-partitioning to determine whether the dependent table partitions can be dropped by verifying the state of their associated master table partition.

Executing the verification scripts can be a two phase process. The Group2 tables depend on the state of the tables in Group1 thus the Group1 verification scripts need to be executed first. So once the Group1 script has been executed and their partitions dropped, the Group2 scripts can be executed to verify their partitions.

As of SOA PS5, the verify scripts are supplemented with row movement scripts (refer C.4)

Executing the Verification Scripts

A verification script is provided for the DBA to determine when to drop a partition and its equi-partitioned dependent table. The verification script is located in RCU_HOME/rcu/integration/soainfra/sql/verify.

1. Connect to the database as SYSDBA privileged account to create the database directory PART_DIR. For example:

```
SQL> create or replace directory PART_DIR as '/.../verify'';
```

2. Grant privilege to the SOA schema owner:

```
SQL> grant read, write on directory PART_DIR to dev_soainfra;
```

3. Edit and execute the soa_exec_verify.sql for Group1 and Group2. Enter the partition names that require verification in the array mySoa_drv_list.

   a. To execute function verify_soa.verify_1, pass 1 as the parameter.

b. To execute function verify_soa_verify_2, pass 2 as the parameter.

5. Review the logs and SQL files generated in the PART_DIR directory.

6. Before dropping the partitioning, ensure to run the purge for the date range of the partitions. The purge script parameters should be:

   - purge_partitioned_component set to false
   - min_creation date and max_creation _date matching the targeted partition range

C.3.3. Partitioning Strategies

C.3.3.1. Complete Partitioning

This approach means that all tables of a specific component are partitioned with the following considerations:

- All the tables for a component must be partitioned or none of them should be partitioned.
  - As an example the BPEL tables can be partitioned without having to partition the other components (Mediator, Workflow, etc). However all tables for BPEL must be partitioned.
- All tables that are partitioned need to be have the same date range and partition name to complete equi-partitioning requirements.
- Mandatory to partition the Fabric top level component tables.

C.3.3.2 Partial Partitioning (Available from PS5)

Partial partitioning facilitates the partitioning of a smaller set of high-volume tables. The requirement to only partitioning a few tables should make partitioning more feasible and attractive for large installations:

1. COMPOSITE_INSTANCE (master)
   a. REFERENCE_INSTANCE(dependent)
2. COMPOSITE_INSTANCE (mandatory)
   a. CUBE_INSTANCE (master)
      i. CUBE_SCOPE(dependent)
3. COMPOSITE_INSTANCE (mandatory)
   a. XML_DOCUMENT
4. COMPOSITE_INSTANCE (mandatory)
   a. MEDIATOR_INSTANCE(master)
      i. MEDIATOR_CASE_INSTANCE(dependent)
5. COMPOSITE_INSTANCE (mandatory)

a. MEDIATOR_PAYLOAD

The following constraints need to be obeyed:

- A dependent table cannot be partitioned unless its master is partitioned
- Partitioning of composite_instance is mandatory
- All tables that are partitioned need to be have the same date range and partition name to complete equi-partitioning requirements

C.4. Row Movement Scripts

C.4.1. Long running transactions

Long running composites are composites which remain open beyond the retention period. This has implications for SOA table partitions as they cannot be dropped until all composites they house are closed. Only a few open composites can prevent the verification scripts from stating that a partition can be dropped. This situation has given rise to the new “row migration scripts”.

The row movement scripts update the partition key of the open rows for all the partitions in the group. In effect, these rows get migrated to a different partition based on the new partition key value allowing the partition to be verified and dropped.

C.4.2. Enabling table row movement

Oracle database allows row movement to be enabled or disabled for partitions. When you create (or alter) a partitioned table, the row movement clause can either ENABLE ROW MOVEMENT or DISABLE ROW MOVEMENT. Row movement will need to be enabled for partitions prior to running the row movement scripts.

C.4.3 Usage

The verify scripts will provide a count of the total open instances and the percentage of open instances within a partition. Based on the percentage (possibly less than 5%) a decision can be made on whether to execute the row-movement procedure. There are two new procedures that perform the row migration.

1. Initiates row movement for group 1 tables.

   Parameters:

   - partition_name: name of the partition on which row movement need to be executed
   - new_partition_date: new date to update the partition key column

   PROCEDURE exec_row_movement_1(partition_name in varchar2,new_partition_date in timestamp);

2. Initiates row movement for group 2 tables

   Parameters:

- partition_name - name of the partition on which row movement need to be executed
- new_partition_date - new date to update the partition key column

PROCEDURE exec_row_movement_2(partition_name in varchar2,new_partition_date in timestamp);

C.4.4 Example sequence of steps for Verify/Row movement scripts

The steps would apply for all scenarios involving partial or complete partitioning

Pre-requisites:

- Oracle Directory PART_DIR needs to be created and the soainfra user should be granted write access to it. All the log and result files will be created within this directory
  - Make an O/S directory where the debug logs will be written:

    Host> mkdir -p /…/partdir
  - Connect to the database as SYSDBA privileged account to create the database directory

    SQL> create or replace directory PART_DIR as '/.../partdir';
  - Grant privilege to the SOA schema owner:

    SQL> grant read, write on directory PART_DIR to dev_soainfra;

Step 1: Execute verify script for Group 1

- Edit soa_exec_verify and include the targeted partition name/partition names in the mySoa_drv_list collection.
  ```
  e.g -    mySoa_drv_list.extend(1);
           mySoa_drv_list(1) := 'P01_2011';
  ```
- Connect to sqlplus as soainfra user and execute soa_exec_verify.sql with 1(one) as the parameter, to the script to execute Group 1 verification.
- Check the log and result files in the PART_DIR directory for the targeted partition.
- If any of the partitions cannot be dropped because of open instances, then evaluate the cost of running the row movement script by looking at the percentage open. If the row movement script is to be executed then go to step 2
- If there are no active instance and the partition can be dropped go to step 4

Step 2: Execute row movement script for Group 1

Select an appropriate partition for the open instance to migrate to and determine its date range. Then execute the row movement procedure for group 1:

DECLARE

```
        new_partition_date timestamp;
        partition_name varchar2(100);
BEGIN
        new_partition_date  := to_timestamp('2012-01-31','YYYY-MM-DD');
        partition_name := 'P01_2011'
       ;verify_soa.exec_row_movement_1(partition_name => partition_name,
                                       new_partition_date => new_partition_date);
END;
```

Note:

- Repeat the row movement for all the partition names that failed verification.
- Once the row movement for all partition names have been executed , truncate the temp tables by executing the following:

```
DECLARE
    BEGIN
      verify_soa.trunc_verify1_temp_tables;
    END;
```

Step 3: Re-verify partitions for Group 1.

- Execute the verify script for Group1 again to verify that the row movement was successful.
- Check the corresponding log and result files in the PART_DIR directory. There should be no open instances with the result file listing ' drop partition ' statements
- **Ensure to run the purge scripts for the date range of the partitions before the partitions are dropped**.

Step 4: Execute purge scripts for non-partitioned tables

- Before dropping the partitions, we need to execute purge for the non-partitioned tables.
- Execute the appropriate purge script with the following parameters:
  - purge_partitioned_component set to false
  - min_creation date and max_creation _date matching the targeted partition range.

Step 5: Drop Group 1 partitions

- Execute the drop partition script (corresponding to the targeted partition) which was created in the PART_DIR directory

Step 6: Execute verify script for Group 2

- Connect to sqlplus as soainfra user and execute soa_exec_verify.sql with 2 (two) as the parameter.

- Check the log and result files generated in the PART_DIR directory corresponding to the targeted partition.
- If any of the partitions cannot be dropped because of open instances, then evaluate the cost of running the row movement script by looking at the percentage open. If the row movement is to be execute then go to step 7
- If there are no active instance and the partition can be dropped go to step 9.

Step 7: Execute row movement script for Group 2

Simlar to step 2, except verify_soa.exec_row_movement_2.

Step 8: Re-verify partitions for Group 2

- Similar to Step 3, except soa_exec_verify pass 2 as the parameter.

Step 9: Drop Group 2 partitions

- Execute the drop partition script (corresponding to the targeted partition) generated in the PART_DIR directory

List Of Tables and their Partition Groups

TABLE 9 COMPONENT: SOA INFRSTRUCTURE (FADRIC)

TABLE	RANGE PARTITION KEY	GROUP
COMPOSITE_INSTANCE	CREATED_TIME.	1
REFERENCE_INSTANCE	CPST_PARTITION_DATE	1
COMPOSITE_INSTANCE_FAULT	CPST_PARTITION_DATE	1
COMPOSITE_SENSOR_VALUE	CPST_PARTITION_DATE	1
COMPONENT_INSTANCE	CPST_PARTITION_DATE	1
REJECTED_MESSAGE	CREATED_TIME	1A
REJECTED_MSG_NATIVE_PAYLOAD	RM_PARTITION_DATE	1A
INSTANCE_PAYLOAD	CREATED_TIME	2
COMPOSITE_INSTANCE_ASSOC	CREATED_TIME	2

TABLE 10 COMPONENT: BPEL

TABLE	RANGE PARTITION KEY	GROUP
CUBE_INSTANCE	CPST_INST_CREATED_TIME.	1
CI_INDEXES	CI_PARTITION_DATE	1
CUBE_SCOPE	CI_PARTITION_DATE	1
DOCUMENT_CI_REF	CI_PARTITION_DATE	1
AUDIT_TRAIL	CI_PARTITION_DATE	1
AUDIT_DETAILS	CI_PARTITION_DATE	1
DLV_SUBSCRIPTION	CI_PARTITION_DATE	1
WORK_ITEM	CI_PARTITION_DATE	1
AUDIT_COUNTER	CI_PARTITION_DATE	1
WI_FAULT	CI_PARTITION_DATE	1
DLV_MESSAGE	RECEIVE_DATE	1A
HEADERS_PROPERTIES	DLV_PARTITION_DATE	1A
DOCUMENT_DLV_MSG_REF	DLV_PARTITION_DATE	1A
XML_DOCUMENT	DOC_PARTITION_DATE	2

TABLE 11 COMPONENT: MEDIATOR

TABLE	RANGE PARTITION KEY	GROUP
MEDIATOR_INSTANCE	COMPOSITE_CREATION_DATE.	1

MEDIATOR_CASE_INSTANCE	MI_PARTITION_DATE	1
MEDIATOR_CASE_DETAIL	MI_PARTITION_DATE	1
MEDIATOR_AUDIT_DOCUMENT	MI_PARTITION_DATE	1
MEDIATOR_DEFERRED_MESSAGE	CREATION_TIME	1A
MEDIATOR_PAYLOAD	CREATION_TIME	2

TABLE 12 COMPONENT: HUMAN WORKFLOW

TABLE	RANGE PARTITION KEY	GROUP
WFTASK	COMPOSITECREATEDTIME	1
WFTask_TL	COMPOSITECREATEDTIME	1
WFTaskHistory	COMPOSITECREATEDTIME	1
WFTaskHistory_TL	COMPOSITECREATEDTIME	1
WFComments	COMPOSITECREATEDTIME	1
WFMessageAttribute	COMPOSITECREATEDTIME	1
WFAttachment	COMPOSITECREATEDTIME	1
WFAssignee	COMPOSITECREATEDTIME	1
WFReviewer	COMPOSITECREATEDTIME	1
WFCollectionTarget	COMPOSITECREATEDTIME	1
WFRoutingSlip	COMPOSITECREATEDTIME	1
WFNotification	COMPOSITECREATEDTIME	1

WFTaskTimer	COMPOSITECREATEDTIME	1
WFTaskError	COMPOSITECREATEDTIME	1
WFHeaderProps	COMPOSITECREATEDTIME	1
WFEvidence	COMPOSITECREATEDTIME	1
WFTaskAssignmentStatistic	COMPOSITECREATEDTIME	1
WFTaskAggregation	COMPOSITECREATEDTIME	1

TABLE 13 COMPONENT: ORACLE BPM SUITE

TABLE	RANGE PARTITION KEY	GROUP
BPM_AUDIT_QUERY	CI_PARTITION_DATE	1
BPM_MEASUREMENT_ACTIONS	CI_PARTITION_DATE	1
BPM_MEASUREMENT_ACTION_EXCEPS	CI_PARTITION_DATE	1
BPM_CUBE_AUDITINSTANCE	CI_PARTITION_DATE	1
BPM_CUBE_TASKPERFORMANCE	CI_PARTITION_DATE	1
BPM_CUBE_PROCESSPERFORMANCE	CI_PARTITION_DATE	1

SOA 11g Database Growth Management
January 2012

Author:
Michael Bousamra
Contributing Authors:
Deepak Arora
Sai Sudarsan Pogaru

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