

The Do's and Don'ts of Space & Undo Management: Best Practices for Oracle Database 10g

An Oracle White Paper

December 2004

The Do's and Don'ts of Space & Undo Management: Best Practices for Oracle Database 10g

Introduction	3
Space Management.....	3
Oracle Basics	3
Fragmentation	5
External Fragmentation.....	5
Internal Fragmentation.....	6
Monitoring Space Usage	10
Resumable Statements	11
Server Generated Tablespace Alerts.....	11
Performance Considerations	12
Segment Space Management	12
Extent Management.....	14
Default Permanent Tablespace.....	17
Temporary Space Management	18
Undo Management.....	21
Manual versus Automatic Undo Management	21
AUM Administrative Practices	22
Undo Retention Configuration	22
Undo Tablespace Configuration	25
Proactive Error Prevention and Automatic Problem Handling.....	27
Conclusion.....	29
Acknowledgements	29
Appendix A: Examples.....	30

The Do's and Don'ts of Space & Undo Management: Best Practices for Oracle Database 10g

INTRODUCTION

Oracle has progressively been expanding its space and undo management solutions with every new database release and it now offers many different features that help enhance space utilization, improve performance, and reduce unwanted exceptions. All of these enhancements have been designed with one ultimate objective: to simplify space and undo management and to make the database administrator's task easier. This paper discusses all the various solutions in the space and undo management area and describes in detail when to use which solution, their most effective use, how each solution works, and ways to avoid common problems.

SPACE MANAGEMENT

Space management is central to any database administration. Databases continue to grow in capacity every year and according to Forrester Research estimates, data growth rate in enterprises exceeds 50% annually. Terabyte databases are not uncommon these days. The number of tablespaces and segments in databases has grown exponentially. Rapid increases in database sizes and concurrency of space management operations have added to the database administrator's space management responsibilities. The primary goals for space administrators are:

- Handling fragmentation and re-organization
- Capacity planning and resource allocation
- Monitoring space usage, and
- Monitoring database performance

This section of the paper discusses space management. It clears many myths about space management and suggests practices to avoid common pitfalls and to best utilize the space management capabilities of the Oracle Database 10g.

Oracle Basics

The database can be seen as a collection of datafiles. The user data and system-generated data reside in entities called schema objects. The schema objects are physically stored on disk as one or more segments. There is one segment per object

usually, unless the object is partitioned or contains a lob column. Clusters can store multiple objects in one segment.

The segments reside in logical collection of files called tablespaces. The tablespaces are of three types depending on the nature of the content they store – temporary, permanent, and undo.

The schema objects map to one or more segments, and each segment is a collection of extents. Extents are contiguous blocks in an ASM/OS file or raw device, not necessarily contiguous on disk. They provide for better I/O performance by increasing the ability to prefetch data. The extents also help keep the metadata compressed. Each extent consists of data blocks. Data blocks are the smallest unit for space allocation.

In Oracle Database 10g you would have the following types of tablespaces:

- User tablespace: Used to store user data. A database can consist of one or more of such permanent tablespaces.
- Temporary tablespace: Used to store temporary data generated during regular database operations. Once again, a database can have one or more of these temporary tablespaces.
- Undo tablespace: Undo tablespaces are used to store the undo data generated by the Oracle Database for transaction rollback and read consistency purposes. A database contains one undo tablespace per instance.
- SYSTEM Tablespace: There is one SYSTEM tablespace per database. It is permanent in content type and contains database dictionary information vital for the functioning of the database.
- SYSAUX Tablespace: Similar to the SYSTEM tablespace, there is one SYSAUX tablespace per database. It is permanent in content type and contains operational data used for self-management purposes, e.g., the Automatic Workload Repository, as well as metadata that is specific to an application, like XDB.

Over the years, Oracle has made several space related enhancements to address wide variety of space management problems. There are a myriad of options available for the user to choose from while configuring and managing disk space. The following sections discuss the common space management problems faced by the administrator and outline Oracle's recommended approach to resolve them. It also spells out the common pitfalls that administrators should be aware of in order to avoid jeopardizing the system integrity.

The hardware configuration used for all the performance benchmark numbers included in this paper is as follows:

- Intel box with 4 CPU, 2.8GHz Xeon processors

- 8GB RAM
- Linux Operating System.

Fragmentation

Fragmentation at a high level refers to inefficient utilization of disk space, which can also result in severe deterioration in application performance. Fragmentation can mean different things in the context of different products. From Oracle's space administration point of view there are three types of fragmentation of disk space:

- **External Fragmentation:** This refers to tablespace level or file level fragmentation. Small pockets of free space are scattered throughout the file which are neither large enough for future allocations nor they can be coalesced together rendering them unusable.
- **Internal Fragmentation:** This refers to the segment level fragmentation where space allocated to the segment is not completely used. The free space in the segment is not available for use by other segments, resulting in space wastage.
- **Object fragmentation:** This refers to the fragmentation of used space in the object due to several small fragments of allocated space spread all over the disk. This can cause performance problems.

Sometimes the storage managers intentionally fragment the data to achieve higher I/O performance. This type of fragmentation will not be discussed in this paper.

External Fragmentation

External fragmentation happens when objects of different sizes are repeatedly created and deleted in the tablespace. The volatility of the segments itself does not lead to fragmentation, it is in part an artifact of the extent allocation scheme. In dictionary managed tablespaces, the extent management scheme allows the users to choose the rate at which extent size should be changed. When the parameters are not chosen properly, it will lead to the creation of extents of several different sizes, which is a recipe for fragmentation. Experiments have shown that the performance of space management operations can degrade 10%-20% when a tablespace is fragmented. Oracle's solution for tablespace fragmentation is to use locally managed tablespaces in place of dictionary managed tablespaces.

Locally Managed Tablespaces

The locally managed tablespaces (LMT) feature was first introduced in Oracle8i. It allows the database to automatically manage the extent sizes and minimizes file or tablespace level fragmentation. There are two flavors of extent management options available with LMTs:

- i. **Auto-allocate Extent Management:** This is Oracle's adaptive extent management scheme where the extent sizes are totally managed by the database. The extent size starts at 64k and is varied progressively in steps that go up to 64M. The size of the extent allocated is based on the segment growth

and the free space available in the tablespace. For medium and large size segments (i.e., segments larger than 1M in size), fragmentation is avoided by making sure that an extent allocated is always a multiple of 1M and when the desired free space is not available in the tablespace, a step down algorithm tries reduces the size of next extent and allocates available space in multiples of 1M thereby minimizing file level fragmentation.

- ii. Uniform Extent Management: In this extent management scheme, all extents are of the same size. The user can choose the uniform extent size at tablespace creation time. External fragmentation does not exist in this tablespace type since any free slot is a candidate for subsequent allocation. The extent size specification cannot be changed once chosen. Uniform tablespaces can be used under the following circumstances:
 - Different segments of the application have approximately the same size.
 - The DBA is able to predict the growth of the segments *a priori*, and expects no surprise in growth rate in the future.
 - Parallel direct loads using slaves are not common.

Best Practice: Use Locally Managed Tablespaces with auto-allocate to avoid external fragmentation of tablespace.

The auto-allocate tablespaces perform equally well in all the above the scenarios. It is the default extent management option for LMTs and Oracle recommends that this option be used.

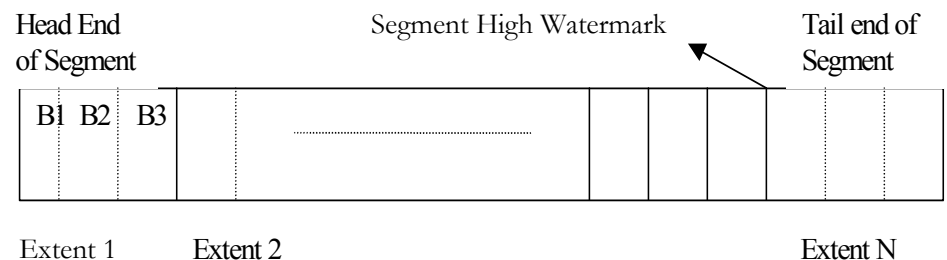
Impact of External Fragmentation On Tablespaces Converted To LMT

Starting Oracle 8.1.7, dictionary managed tablespaces can be converted to locally managed tablespaces using the DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_TO_LOCAL procedure. This procedure changes the extent space management policy for the tablespace from being dictionary to locally managed. No physical changes are however made to existing extents and the advantages of local management will only accrue to the newly created objects. Therefore, if fragmentation was a concern before the conversion, it will continue to remain so after the conversion. In such cases, Oracle recommends an out-of-place migration of data by creating a new locally managed tablespace and moving data into it using the online table reorganization feature.

Best Practice: If dictionary managed tablespace is already fragmented, then do not do in-place migration to locally managed type. Create a new locally managed tablespace and move data into it.

Internal Fragmentation

Oracle segment can be seen as a linear array of data blocks. High watermark (HWM) is the pointer to the data block below which all blocks are considered used and above which all blocks are unused.



Fragmentation of the segment space, or under utilization of the data blocks under the HWM is referred to as internal fragmentation in Oracle. Segment fragmentation is a serious issue for two main reasons: 1) poor space utilization, i.e., the free space under the HWM of one segment cannot be reused by other segments without doing shrink first, and 2) poor performance, where certain data access patterns are impacted due to having to do more I/O than is necessary to read the data. Fragmentation can impact heap, index as well as lob segments.

Fragmentation of Heap and Index Segments

The following reasons can cause index or heap organized segments to get fragmented:

1. Incorrect setting of PCTFREE: PCTFREE is a segment storage parameter used to specify how much space should be reserved in the blocks for future updates that cause rows to grow. Once the free space of the block falls below PCTFREE, the block is considered FULL, and future inserts cannot happen in the block. If the application is mostly insert only or the updates do not change the length of the rows in proportion to the reserved space, then the reserved space will remain unused.
2. Incorrect setting of PCTUSED: PCTUSED is another segment storage parameter applicable only to heap segments. It is used to specify as to when a block previously marked FULL should be considered for new inserts. When the used space in the block goes below this value, the block is considered a candidate for inserts. If the value of PCTUSED is too low and sufficient deletes do not happen then the block will remain unusable for future inserts causing sparseness in the segment
3. Queue like behavior with temporal data in heap segments: This is usually seen in data warehouse applications where data is typically direct loaded above the segment HWM. After the data 'expires' it is deleted from the table. Repeating this cycle causes severe fragmentation of the segment since all the new inserts happen above segment HWM and the space below the HWM never gets reused.
4. Staging Tables: Such tables are used to temporarily store data, acquired from external sources, so that they can be transformed and massaged into the desired format. Bulk of the data loaded into the table is deleted once the transformation is complete. Such frequent and large delete operation can also cause segment level fragmentation.
5. Index segments with random updates and deletes can end up with index leaf blocks that have a lot of free space in them.

Fragmentation of Lobs

The lob data is maintained in a separate segment of its own. Lob segment fragmentation can happen when we have:

1. Incorrect setting of PCTVERSION/RETENTION: The undo for changes made to a lob segment is kept in the lob segment itself. The PCTVERSION parameter is used to control the percentage of the lob segment that must be allocated to retain the version data. Alternatively RETENTION parameter is used to determine for how long (in seconds) the lob undo should be retained. If the value of PCTVERSION or RETENTION was set to a very large number and if the queries on the lob segment are relatively shorter, then a large portion of the segment could be wasted in retaining the older lob versions.

Performance Impact of Internal Fragmentation

In a heap and index organized segment, fragmentation impacts most of the segment access paths. Full table scan of the segment will have to process all the blocks under the HWM whether they contain useful data or not. The overhead involved is in issuing extra I/Os plus performing consistent reads on the extra blocks. Similarly, fast full scans on the indexes, range scans on the heap segments and DMLs that involve the above scans will show an increased response time due to sparseness in the segments. Loss of data clustering also impacts the efficiency of buffer cache by decreasing the cache hit ratio. In the case of lobs, data in such segments is always accessed through an index. Hence there is no impact in access performance due to fragmentation for lobs.

The following table shows the impact of fragmentation on an 800 MB heap organized table with various degree of fragmentation. The access path used is full table scan executed in a single thread. The table without fragmentation is a fully compacted segment.

Table 1

Fragmentation	Scan Time Without Fragmentation	Scan Time With Fragmentation
20%	8.53s	26.17s
50%	2.47s	26.33s
70%	0.08s	26.67s

Note that fragmentation impacts the scan time by several orders of magnitude.

Internal Fragmentation Remedy

Fragmentation is an age-old issue that the DBAs have been battling for a long time. Use of locally managed tablespaces in conjunction with Automatic Segment Space Management (ASSM) offers the best way to minimize internal fragmentation. ASSM manages free space within a segment by using bitmaps instead of freelists. In this new scheme, a set of bits describes the space utilization for each block in a segment as well as whether it is formatted or not. Unlike FREELISTS which just have 2 states which show whether a block is available for insert of new rows or not, bitmaps provide a more granular and accurate picture of space utilization within

blocks. This allows for better space utilization of the data blocks and thereby helps reduce internal fragmentation. ASSM also has a direct impact on the performance of transactions. We will defer a more detailed discussion on ASSM to a later section where we discuss the performance considerations of space management in more detail.

Once internal fragmentation has occurred, the traditional methods adopted by the DBAs to solve the problem includes:

- Alter table MOVE, CTAS (create table as select), Import/Export.
- Reorganizing the object using online redefinition set of procedures
- Use of online rebuild for index segments

The above techniques are effective in removing fragmentation, but require additional disk space and involve considerable manual labor. In Oracle Database 10g, Oracle introduced the more specialized *Online Segment Shrink* feature targeted towards eliminating fragmentation in segments. The following table summarizes the benefit of each approach and compares them (with respect to fragmentation).

Table 2

Property	Online Segment Shrink	Online Redefinition	ALTER ... MOVE
Online	Y	Y	N
In-place	Y	N	N
Incremental	Y	N	N
Automatic Dependency Maintenance	Y	N	N
Segment Level Reorg	Y	N	Y
Parallel	N	Y	Y

Online Segment Shrink works on heaps (tables), indexes and lob segments. Shrink can be done on an entire table or a single segment. Additionally, shrink can be cascaded to all the dependent segments. Shrink is done by moving the rows in the segment in such a manner that all the free space is aligned towards the end of the segment. The HWM is then lowered and free space released. The data movement is done in multiple small transactions and the progress is registered. Hence shrink works incrementally. Online Segment Shrink works on all segments created in tablespaces with Automatic Segment Space Management.

Best Practice: Use Online Segment Shrink to eliminate internal fragmentation

When to Defragment Segment

A fragmented segment is not necessarily a cause for concern. If a terabyte segment has few MBs free, then it makes less sense to defragment the segment. Similarly, if the access path is for example always index based in a heap segment, then there is no performance need to defragment the heap segment. DBAs often setup scripts that run periodically to analyze the space usage and performance impact and schedule defragmentation. In Oracle Database 10g, the Segment Advisor was introduced. This advisor is capable of detecting fragmentation more efficiently and

Best Practice: Use Segment Advisor periodically to identify segments needing defragmentation. It is not necessary to defragment all fragmented segments.

makes a recommendation to rectify the problem. The Oracle Database 10g periodically collects space usage information for each segment and stores it in the Automatic Workload Repository (AWR). The segment advisor uses the space usage history information to determine the growth trend of the segments and to detect if shrinking or reorganizing a segment can result in any significant space reclamation. The recommendations and findings are can be viewed using Enterprise Manager (EM) or through DBA_ADVISOR family of views.

Automatic Segment Advisor

Better yet, in Oracle Database 10g Release 2, the Segment Advisor is automated and comes enabled out-of-the-box. The advisor is run in the default maintenance window, which is open during nights on weekdays and all the time on weekends. The Automatic Segment Advisor examines segments residing in tablespaces with outstanding tablespace-full alerts and those with heavy user access for signs of fragmentation. If the segments are fragmented, a recommendation is made to rectify the problem. The recommendation could be an advice to shrink the segment if it is a candidate for segment shrink or for online redefinition if segment shrink is not permitted. Note that the Automatic Segment Advisor does not automatically implement its recommendations. DBA's can review the recommendations and implement them as they see fit.

The results of the advisor are available through the usual advisor framework tables (command-line interface) and can also be viewed using Oracle Enterprise Manager graphical interface. The DBA must periodically review the results of the advisor and schedule defrag of concerned objects. The Automatic Segment Advisor generates recommendations for both automatic segment space management and manual segment space management segments, so as to eliminate the need to setup handcrafted scripts.

Best Practice: Periodically review the results of Automatic Segment Advisor.

Monitoring Space Usage

Historically, about 20% of all technical requests logged with product support relating to space management in prior releases were associated with external errors raised due to improper configuration of the database. A majority of the errors reported relate to out-of-space errors. The reasons for this high failure rate are due to 1) the inability of the DBA to predict the space usage patterns, either because the database and application are new or because there is no predictability in their space usage, 2) the number of tablespaces and segments are so large that it is impossible to pay attention to each one of them, and 3) improper configuration of tablespace quotas for the users resulting in runaway DMLs going unchecked. The following space management limits, when hit can cause applications to fail.

- **Tablespace out of space:** When the tablespace runs out of space, it adds more space from the file system if the files are autoextensible. If the files are not autoextensible or if the file system is full, then out-of-space error is raised.

- Segment MAXEXTENTS limit hit: With dictionary managed tablespaces, the maximum number of extents in a segment is configurable through MAXEXTENTS parameter. When this limit is hit, an error is raised. This is not an issue with locally managed tablespaces since the limit is set to UNLIMITED and is not configurable.
- User Quota limit reached: The user can associate quotas with permanent tablespaces. This is the maximum space that the user can consume in the tablespace. When it is exceeded, space-quota-exceeded error is raised.

Oracle provides a complete solution to deal with the space limit errors. Resumable statements and proactive alerting of impending problems help DBAs in avoiding configuration problems.

Resumable Statements

The resumable statements feature allows the session that runs into a space management error to be suspended, so that the DBA can take corrective actions. This is especially useful if the operations are long running batch jobs or queries and space usage could not be predicted ahead of time. The space management errors handled by resumable sessions include 1) Out-of-space errors 2) MAXEXTENTS reached errors 3) User Quota exceeded errors. The statements remain suspended for a configurable timeout period, which by default is two hours. The details of the suspended statements can be seen through the {DBA|USER_RESUMABLE} views. In Oracle Database 10g, an operation suspended alert is issued on the statement that gets suspended due to out-of-space error.

Best Practice: For long running statements that consume space, if space usage cannot be predicted, enable resumable session.

Server Generated Tablespace Alerts

Oracle Enterprise Manager (OEM) provides the ability to monitor the space limits and raise an alert when the limit is hit. Warning and Critical thresholds can be set on the alert metric. When the metric value, for example, the fullness of the tablespace crosses the threshold limits, an alert is raised. The EM polls the database for free space information periodically and evaluate the thresholds. This alert infrastructure was only available through OEM until Oracle Database 10g. In Oracle Database 10g, the tablespace alerts are integrated into the database server. The server internally keeps track of the space usage information in the SGA. A special background process called the MMON evaluates the tablespace space usage information once every 10 minutes to raise an alert. All the metadata needed to raise and maintain the alert status is kept in the database server. The evaluation of alerts is therefore much more efficient. The tablespace alerts can be viewed using regular dictionary views:

```
SQL> SELECT DECODE(MESSAGE_LEVEL, 5, 'WARNING', 1,
                  'CRITICAL') ALERT_LEVEL, REASON FROM
        DBA_OUTSTANDING_ALERTS WHERE OBJECT_NAME = 'PERM';
```

ALERT_LEVEL	REASON
CRITICAL	Tablespace [PERM] is [91 percent] full

Best Practice: Monitor tablespace space usage using server generated alerts for impending out-of-space conditions.

The tablespace alerts are enabled out-of-the-box. A warning threshold of 85% and critical threshold of 97% are set on all the tablespaces. The DBA can either monitor the dictionary views or use OEM to look out for problems.

Performance Considerations

The choice of space management attributes at tablespace and segment level can impact the performance of the database. OLTP transactions can bottleneck on buffer busy waits on space metadata blocks. DSS and OLAP applications can have increased response times due to improper choice of extent management in temporary tablespace. The extent management scheme can impact the response time of parallel queries. Oracle traditionally provided control to the user in the form of storage parameters that is specifiable at the tablespace and segment level. With the increase in size of databases and number of tablespaces and segments it is not possible to tune the storage parameters at segment level. Also wrong choice of storage parameters can adversely impact the performance of applications. The factors that may influence the performance of database space management performance include:

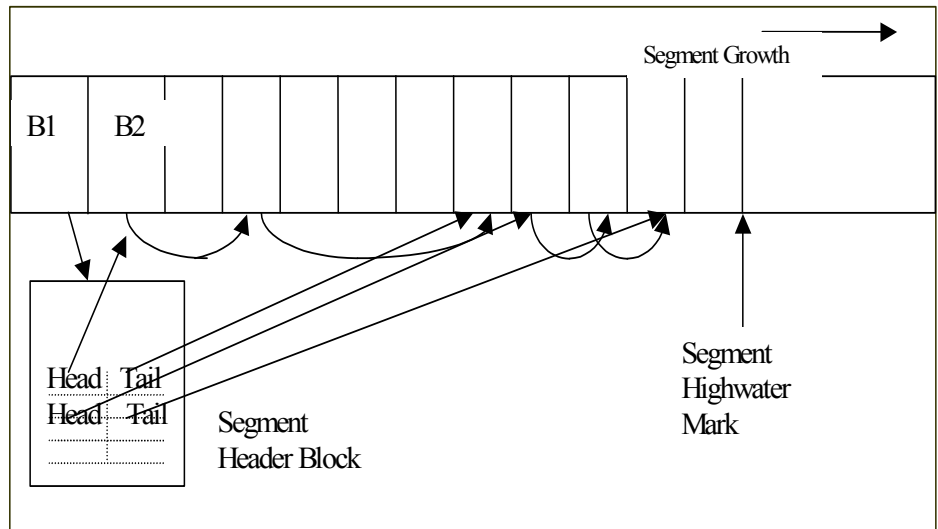
- Segment space management
- Extent space management
- Number and size of extents in the segments and tablespaces

The influence of some of the above factors on performance is commonly misunderstood. In the following sections each of the above is discussed:

Segment Space Management

Segment space management involves the management of free space at the block level. Data structures used to represent the free space within a segment are of two types: freelists and bitmaps. Prior to Oracle9i, freelists was the only way to represent the free space. In freelist space management blocks that have space for future inserts are kept in a linked list, the head and tail DBAs (data block address) of which were kept in the segment header block.

The following figure shows the structure of a freelist -managed segment. B1, B2 are database blocks. The first block in the segment is the segment header block that contains along with other information, the freelist metadata. The head and tail DBA of the linked list of free blocks is kept in the segment header block. Processes trying to allocate space will use the blocks at the head of the freelist. PCTFREE and PCTUSED parameters control when the block should be removed from the freelist and put back in the freelist.



FREELISTS Storage Specification: The presence of single freelist for the entire segment causes serious contention on the data blocks at the head of the freelist. FREELISTS storage parameter allows the user to increase the number of insert points in the segment. While FREELISTS reduces the number of buffer busy waits on the data blocks at the head of the freelists, it can cause wastage of space and still leave the segment header block heavily contended for. To ease the contention on segment header block, FREELIST GROUPS was introduced, which allowed multiple blocks to hold the freelist metadata.

FREELIST GROUPS reduce the metadata contention, but it has the undesirable side effect that it statically partitions the segment. The data blocks freed by one instance cannot be used in other instances. Reconfiguration of FREELIST GROUPS parameter cannot be done without rebuilding the object making it an unmanageable storage parameter.

Automatic Segment Space Management (ASSM)

Oracle's solution to segment space management performance and manageability is Automatic Segment Space Management, which was introduced in Oracle 9i. In this segment management type, the free space in the segment is managed using bitmaps. The free space of each data block is represented by a set of bits. Contention on metadata blocks is reduced by keeping the metadata in several metadata blocks. Since the free space information of the entire segment is exposed through a set of bitmap blocks, the free space allocation algorithms can search for space more effectively. Buffer busy waits on data as well as metadata blocks is negligible in ASSM segments.

In Real Application Cluster (RAC) environments, contention on metadata blocks as well as data blocks is reduced by having dynamic instance affinity. An instance retains affinity over the bitmap blocks as long as it actively allocates free space from it. When an instance is no longer using the bitmaps, other instances can transfer the

ownership. When instances are added or deleted from the cluster the bitmap ownership balances transparently and dynamically on demand basis.

Best Practice: Oracle recommends that you always use Automatic Segment Space Management for permanent tablespaces storing user data. It performs better, it scales better and it provides superior space management.

The following charts (Figure 1) show the execution time of ASSM in comparison to manual segment space management for insert only workload and mixed DML workload. The amount of data in the segment is 128M. The mixed DML workload consists of 50% inserts and 50% deletes.

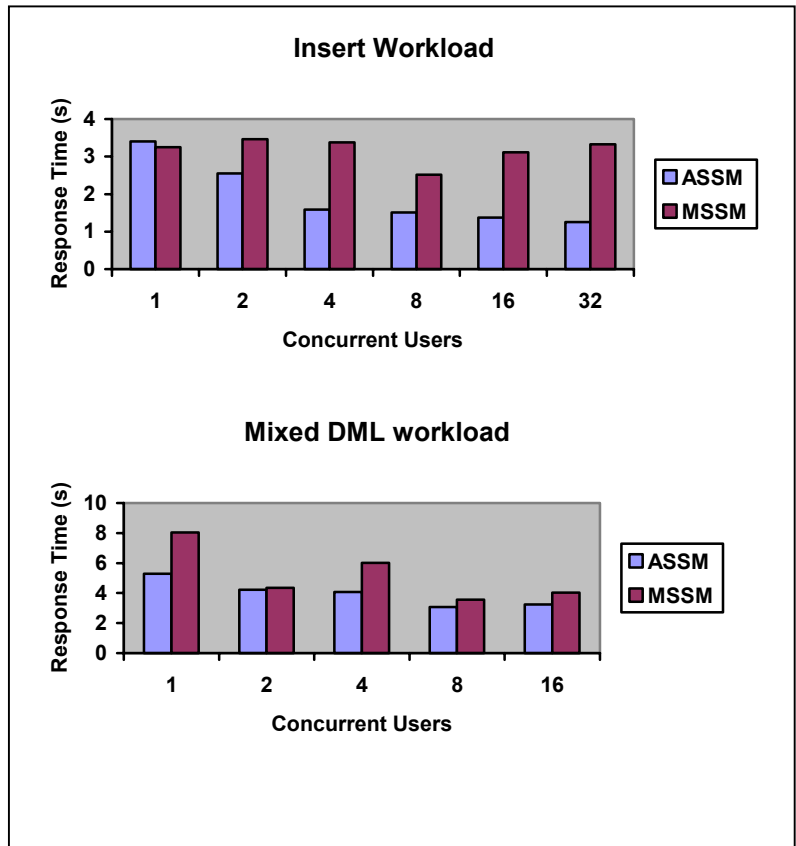


Figure 1

The total amount of data in the workload is kept constant. Note that the response time for ASSM remains well under the response time for MSSM. ASSM is not an option (and is not necessary) for segments in undo and temp tablespaces.

Extent Management

The extent space management in dictionary managed tablespace is sub-optimal when compared to locally managed tablespaces. This is attributed to three main reasons.

- Database level serialization through ST enqueue: The extent management operations like extent allocation and deallocation involve updating several dictionary tables. Since the dictionary tables are centralized in the database, and updates to the tables have to happen in certain order, the operation is serialized at the database level through the ST enqueue. This seriously impairs the performance of DMLs and DDLs in applications that are space management intensive.
- Fragmentation: Dictionary managed tablespaces allow extent sizes ranging from 2 data blocks to two gigabytes. The user has fine control over the extent sizes in the form of specifying INITIAL, NEXT and PCTINCREASE storage parameters. When the parameters are not carefully chosen for each segment, it increases the number of different sized extents in the tablespace, which is a recipe for tablespace fragmentation. The CPU and I/O time spent in searching for right sized extent increases highly in a fragmented tablespace.
- Metadata maintained in dictionary tables: In dictionary managed tablespace, the metadata is maintained in dictionary tables (UET\$ - used extents, FET\$ - free extents, FILE\$ - file, TSQ\$ - tablespace quotas, SEG\$ - segment). Every space management operation will involve updating at least three dictionary tables (UET\$, SEG\$ and FET\$) and can involve updating all the five dictionary tables if user has quotas (TSQ\$) and file has to be extended (FILE\$). The recursive SQLs can trigger recursive space management and can involve update of more dictionary tables.

The locally managed tablespaces (LMT) manage free space using bitmaps. Each bit in the file maps one allocation unit. The allocation unit is an extent in uniform tablespace and 64k in auto-allocate tablespace. Allocating and deallocating extents involve flipping one or more bits in addition to updating a few dictionary tables. Also there is no requirement to split or coalesce free space in the bit stream as the bits in the bitmaps are auto-coalesced. Extent allocation and deallocation operations are 100%-200% faster in locally managed tablespaces. Since the serialization in locally managed tablespace is at the file level rather than the database level, concurrency of space management is far superior in locally managed tablespaces.

Conversion from Dictionary Managed to Locally Managed Tablespace

Databases that are still using dictionary managed tablespaces can be converted to locally managed tablespaces. The tablespace conversion can be done by using the DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_TO_LOCAL procedure. Migration is done in-place and online, and multiple tablespaces can be migrated in parallel.

Best Practice: Use locally managed tablespaces for better performance. Convert the existing dictionary managed tablespaces to locally managed type using online, in-place migration procedure DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_TO_LOCAL.

Number and Size of Extents

The list of extents allocated to a segment is stored in one more metadata blocks. The extent map is read by queries taking access paths like full table scans, fast full scans, and range scans. DDLs like truncate, segment shrink, drop will read the extent map to delete the entries. There is a common notion that fewer extents in the segment show better performance than multiple extents. Some administrators make sure that the entire segment fits in one extent. The following table shows the number of metadata blocks required to contain the extent information with respect to 4k, 8k and 16k block sizes in an ASSM tablespace with auto-allocate option.

Table 3

Tablespace Block Size	Segment Size		
	100M	1G	100G
4K	1	2	8
8K	1	1	4
16K	1	1	2

Given that the addressability of each metadata block is huge, the time spent in reading a metadata block is negligible when compared to the time spent in reading the actual data.

There are however cases where the number of extents should be a cause for concern:

- Tablespace is dictionary managed (DMT) and DDLs like truncate and drop are common: In DMTs, each extent has a representation in the data dictionary tables. The DDLs like truncate and drop of segments will have to update the dictionary data, which is an expensive operation in terms of CPU and I/O. For segments that have high volatility and performance of DDLs is critical to the database, a reasonable number of extents to have that will not impact the performance is 1024 to 4096. Choosing different extent configuration is recommended for such segments. This is not a concern in locally managed tablespaces. The following table shows the impact on DDLs in dictionary managed versus locally managed tablespace. The value in the table indicates the response time for dropping a table in single instance with as many extents.

Table 4

Tablespace Type	Number of Extents		
	100	1000	10000
LMT	0.01s	0.74s	29s
DMT	0.24s	6.36s	61s

- Parallel queries are common and extents are noncontiguous in file: The parallel query coordinator reads the extent maps, collapses contiguous extents into larger chunks and constructs the range of DBAs (data block address) to be processed by each parallel query slave. If the number of extents are in the range of millions and most of the extents are not coalescible, the performance of the query coordinator can be impacted. In this case, Oracle recommends either partitioning the segment or using larger sized uniform extents. Note that as of Oracle Database 10g Release 1, the largest auto-allocate extent size possible is 64M. This is a problem conceived in segments in size close to terabytes and more and should not impact segments of lesser sizes.

The number of extents in any other case should not be a cause for concern. What matters more is the size of the extent. The administrators should make sure that the extents are reasonably sized when they create uniform tablespaces. When sizing extents, take the following into consideration:

- I/O performance: Very small extent sizes do not allow for multi-block reads or prefetch of data. It can impact the read as well as write performance of the segment. Depending on the access pattern of a particular segment, and the size and access patterns of the other segments created in the tablespace, the uniform extent size should be chosen. The general guideline is to use 64k for tablespace containing predominantly small segments, 1M when tablespace contains medium sized segments and 100M for very large segments.
- Avoid large sized extents during segment merge loads: Segment merge load is a type of parallel direct load in which each slave creates its own private temporary segment, loads data into it and then merge it to the base segment. Segment merge load is used when a user invokes CREATE TABLE AS SELECT or CREATE INDEX in parallel or with parallel insert direct load operations that does not use high watermark brokering. During segment merge load, the last extent of the temporary segment that is merged to the base may contain unused blocks and will get merged as it is. Unless the segment is subsequently loaded using conventional inserts, these blocks will remain unused in the segment. Besides space wastage, this also has a side effect that the queries will perform extra consistent reads on these blocks thereby adversely impacting the query execution time. In these cases, the DBA should use auto-allocate extent management. With auto-allocate extent management, the last extent of the temporary segment is trimmed before merge. Hence the space wastage is minimized.

Best Practice: The size of the extents matter more than the number of extents in the segments. Choose uniform extent sizes carefully.

Default Permanent Tablespace

The Oracle database has one permanent tablespace called SYSTEM, which stores the data dictionary information. The SYSTEM tablespace is critical for the

functioning of the database and can be a single point of failure if things go wrong with it. SYSTEM tablespace is also the default tablespace for creating user's permanent and temporary objects. Oracle recommends that SYSTEM tablespace should not be used to store the user permanent objects. This is due to the following reasons:

- Security: A malicious user who is assigned to the SYSTEM tablespace can fill up the SYSTEM tablespace and make the database unavailable by making DMLs and DDLs on dictionary tables fail.
- Performance: The presence of unnecessary objects in the SYSTEM tablespace can impact the performance of those operations that happen at file or tablespace level.

Best Practice: Designate a permanent tablespace other than SYSTEM and SYSAUX as database default permanent tablespace.

Most of the DBAs are aware of this problem, so they create an alternative permanent tablespace and reassign the newly created users to the new tablespace. In Oracle Database 10g, the notion of default permanent tablespace was introduced, whereby a tablespace can be designated as the database default for permanent objects. At the time of database creation, the DBA can specify a default permanent tablespace.

Temporary Space Management

Temporary tablespace or the database scratch space is used to store transient data generated explicitly by the user and implicitly by the system. The data stored in temporary tablespace are predominantly from hash join, sort, bitmap merges, bitmap index creation operations as well as temporary lob and global temporary tables. In DSS and OLAP environments, since the efficient management of transient data is central to the execution of queries, the performance of temporary tablespace is extremely critical. The following guidelines can be used to setup the database for temporary space management

Locally Managed Temporary Tablespace

Depending on the content of the data stored, the tablespaces are classified as temporary and permanent. The space management in temporary tablespaces differs from permanent tablespaces in the following ways:

- The temporary tablespaces contain temp files rather than datafiles that constitute the permanent tablespace.
- Temp tablespace allows high concurrency space management. In steady state, all the space metadata is cached in the SGA. This allows for completely in-memory space management operations. The space management operations in permanent tablespaces used for storing temporary data is serialized by database wide ST enqueue whereas in locally managed temporary tablespace, it is serialized by instance specific SGA latches held for a very short duration.

Best Practice: Use locally managed temporary tablespace for managing transient data generated by the database.

- Redo logging: The redo generated on temp files are automatically discarded after modifying the block. They are not written to the disk. This allows for faster writes to temp files.
- Read-Only Database: All the metadata required by the temporary tablespace is stored in the tablespace itself. This implies that it does not require modification of any files outside the temporary files itself, thereby making read only databases possible.

Due to the inherent performance benefits of locally managed temporary tablespaces, Oracle recommends that they be used in place of dictionary managed temporary tablespaces.

Configuring Temporary Tablespace

While creating temp space for the database, the following attributes are important

1. Tablespace Extent Size

The size of the extent impacts the performance of temp tablespace. The extent size can vary anywhere from 2 data blocks to over 2G. The choice of right extent size involves the usual space versus time tradeoff. Very small extent sizes will impact read/write performance as well as increase the number of space management operations on large segments. Very large extent sizes can cause space to be wasted in the last extent allocated to the segment, with no improvement in performance. First it is important to understand the types of workload on the temp tablespace.

- DSS: The DSS workload is typically dominated by complex queries performing intensive sort and hash join operations. The work areas used by these operations are allocated from the PGA. When the data exceeds the work area size, temp space will be allocated in the disk. Since the data in the work area is written to the temporary segments in multiples of 64k, it is advisable to choose an extent size that is multiple of 64k. The general rule for good performance and efficient space usage for temp workload dominated by DSS queries is to set extent size of 1M.
- Global Temporary Tables: These are transient tables created by the user as well as system in the temporary tablespace. The temp tables usually exist for the duration of transaction or session. Each global temp table requires at least one extent to be allocated. If the volume of data loaded into the temp tables is pretty small, then choosing large extent sizes can cause considerable space wastage. In this case smaller multiples of 64k should be chosen for extent size to avoid space wastage.
- Temporary Lobs: Temporary lobs are created explicitly by the user as well as implicitly by the database to store transient unstructured data. If the temp space usage is dominated by large temporary lobs, then larger extent sizes should be used. Oracle recommends that 1M to 10M is an appropriate extent size for a workload dominated by temporary lobs.

Best Practice: Use 1M to 10M extent sizes for workloads dominated by DSS, OLAP queries and large temporary lobs. If the workload is dominated by OLTP or small temporary tables, use smaller multiples of 64k.

The distribution of workload can be monitored through v\$tempseg_usage dictionary view.

```
SQL> SELECT session_num, username, segtype, blocks,
tablespace FROM v$tempseg_usage;
```

SESSION_NUM	USERNAME	SEGTYPE	BLOCKS	TABLESPACE
101	SCOTT	SORT	128	TEMP
102	SCOTT	LOB_DATA	128	TEMP
103	SYS	SORT	256	TEMP
104	BLAKE	LOB_DATA	128	TEMP

The segtype column indicates what type of segment is using the space. It is one of SORT, HASH, INDEX, LOB_DATA and DATA. Based on the distribution of segment sizes, an appropriate extent size can be chosen.

2. Temporary Tablespace Setup in RAC

Temporary tablespaces are designed to work on the entire database. In RAC environments extents have affinity to instances. Each instance caches the extents it has affinity to in its SGA. Extent allocations and deallocations happen in each instance's extent cache. When one instance is under space pressure, it will steal free space from other instances. This is done by requesting the SMON in other instances to make its unused space available.

Sharing is a relatively expensive operation that should be avoided in steady state. The user can monitor the stealing of space between instances by checking the waits on SS enqueue. If the enqueue wait tops, then the DBA should increase the space available for the temp tablespace.

3. Choice of Single Temporary Tablespace or Temporary Tablespace Group

Temporary tablespace groups were introduced in Oracle Database 10g. It allows the user to create one or more temporary tablespaces and assign them to a group. The entire tablespace group can be designated as the database default temporary tablespace group. Similarly a tablespace group can be designated as the default for a user. The primary purpose of a temporary tablespace group is to increase the addressability of the temp space. A single temporary tablespace can have a maximum of 4 billion blocks. This is 8T for 2k block size and 64T for 16k block size. With temporary tablespace group, the addressability is increased to several petabytes.

When a parallel statement is executed, the temporary tablespace group allows the slave processes to use multiple temporary tablespaces. Temporary tablespace groups should not be used under the assumption that a single temporary segment will span multiple tablespaces.

Best Practice: Use temporary tablespace group if the addressability of a single temporary tablespace of several terabytes is not sufficient.

UNDO MANAGEMENT

One of Oracle Database's greatest strengths over other database products is multi-version read consistency. It is this technology that enables Oracle to have reads and writes not block one another (Oracle does not acquire read locks). This affords Oracle the ability to have unmatched scalability and performance, and allows it to support systems with very high concurrency. Undo segments are key to making all of this possible for the Oracle Database. They maintain information needed to retract or "undo" changes made by uncommitted transactions and make it possible for queries to display completely consistent data as of a particular point in time without requiring any read locks or necessitating "dirty" reads, i.e., reading uncommitted data. Whereas, database administrators using other database vendor products have to concern themselves with configuring lock isolation levels, managing lock escalations and preventing artificial deadlocks, Oracle administrators have no such concerns to complicate their tasks.

In addition, multi-version read consistency is also the key infrastructure that affords Oracle the ability to provide the Flashback capabilities to run queries at older points in time, selectively query older versions of a given set of rows, recover individual tables to certain points in time, etc.

Undo segments store the data necessary to support this infrastructure. Undo segments are persistent first class objects and survive database systems crashes and shutdowns. In this section, we will discuss the best practices to avoid the most common problems associated with Undo management.

Manual versus Automatic Undo Management

Automatic Undo Management (AUM) was introduced in Oracle9i. Users now have the option to continue to use manual undo management using rollback segments or to use AUM. AUM is the preferred option as it greatly simplifies Undo management and also provides an easy way to avoid errors like ORA-1555 (snapshot tool old). Automatic Undo Management enables the database server to automatically manage all aspects of undo segment management. Space allocation, determination of number of undo segments to support concurrency needs are all handled automatically by the database. In contrast, manual undo management requires that the DBA appropriately determine and create number of rollback segments required, ascertain how much space is needed, set extent sizing parameters of individual rollback segments correctly, and ensure that large transactions do not fail by associating them with rollback segments of sufficient size. Whereas all of this is doable, it is by no means trivial. AUM solves this headache for the DBA by having the database itself manage its undo. AUM is also dynamic in that it adjusts the number of undo segments to meet the current workload requirement. It creates additional undo segments whenever required. At the same time undo segments are also brought online and off-line as needed and the space used by them reclaimed whenever they are no longer needed due to reduced workload concurrency. All these operations occur behind the scene with

Best Practice: Use Automatic Undo Management (AUM) instead of manual management via rollback segments.

no intervention by the administrator. With all the benefits that AUM offers, we highly recommend that DBA's choose this mode of undo management over manual management via rollback segments. In the remainder of the paper, we will restrict the focus of undo management best practices to AUM exclusively.

AUM Administrative Practices

To enable Automatic Undo Management, you must create an Undo tablespace, set initialization parameter `UNDO_MANAGEMENT=AUTO` and depending on the release of the Oracle Database set initialization parameter `UNDO_RETENTION` accordingly. Because the behavior of undo retention has changed between Oracle9i, Oracle Database 10g Release 1, and Oracle Database 10g Release 2, we will discuss this parameter first.

Undo Retention Configuration

Overview

Before we start describing how to properly set `UNDO_RETENTION` parameter, a brief discussion on how undo segments work is warranted. Although the Oracle database treats undo as a first class database object, which can be retained forever, it ceases to be of interest after a period of time. Undo information is said to be active if the transaction that generated the undo is still *active* (uncommitted), otherwise it is said to be *committed*. *Active* undo in the system is always preserved while *committed* undo is further categorized into *expired* and *unexpired* undo. Whether committed undo space is considered expired or unexpired depends on the configuration of the initialization parameter `UNDO_RETENTION`. This parameter provides a way to explicitly control reuse of the *committed* undo space. It allows the DBA to specify the amount of time for which committed undo information is retained in the database as unexpired so as to successfully allow read consistency and Flashback operations. Once the time specified by the parameter lapses, the undo space is considered as expired. Now this space is fair game to be used by any new transactions needing to store undo data. In contrast, Oracle will still try to retain unexpired undo for as long as possible unless it is under space pressure and the Undo tablespace has no more room for storing new undo data generated. In this situation Oracle will overwrite unexpired undo instead of failing the current transaction. Note that the oldest unexpired extents are used first in order to minimize their impact on queries and Flashback features. Depending on whether unexpired undo is reused of the same or a different segment, it is called undo *reuse* or *stealing*. Stealing and reuse are both evidence of the Undo tablespace being too small or undo retention setting being too large, with the former being the more common situation. Oracle internally tracks the number of times unexpired undo is reused and stolen in the `V$UNDOSTAT` view:

- `UNXPSTEALCNT`: This column shows the number of times a transaction attempted to obtain undo space by stealing unexpired extents from *other* undo segments.

- UNXPBLKREUCNT: This column shows the number of unexpired blocks reused by transactions from the *same* segment.

The only instance where unexpired undo is *never* overwritten and is treated like active undo is when the RETENTION GUARANTEE clause for the undo tablespace is set. This ensures that *unexpired* undo is not discarded even under space pressure from freshly generated undo. You should carefully consider the implication of setting this clause as it can cause other transactions to fail because of lack of space in the Undo tablespace.

Setting Undo Retention

Best Practice: In Oracle9i, set UNDO_RETENTION to at least the value of the longest running query.

As mentioned earlier, the behavior of the UNDO_RETENTION parameter has changed in the last couple of releases. In Oracle9i, this parameter specified the time for which undo was considered unexpired. Hence, the recommended setting for this parameter was to set it to a value that was at least equal to the length of longest running query on a given database instance. If you are not sure what that value is, you can query the V\$UNDOSTAT view as shown below to determine its precise value:

```
SQL> select max(maxquerylen) from v$undostat;

MAX(MAXQUERYLEN)
-----
202
```

This parameter is also used to allow predictable use of the Flashback Query feature of Oracle9i.

In Oracle Database 10g Release 1, undo retention was made auto-tuning. Oracle automatically learns how long to maintain the committed undo data for read consistency purposes. This in effect makes the system self-learning, as the undo retention is auto-tuned to match application needs. If your system workload varies between day and night from OLTP to batch/reporting workload, undo retention will adjust downwards during the day to allow for storing more undo data generated by OLTP transaction without putting space pressure on the tablespace, and at night as longer queries are run in the batch/reporting workload, undo retention will adjust upwards to ensure that queries do not fail with ORA-1555 (snapshot too old) error. This is made possible in Oracle Database 10g by tracking all active queries and keeping the undo retention ahead of the maximum query length currently in the system. Now that undo retention is auto-tuning, the purpose and behavior of the parameter UNDO_RETENTION have been modified. It now represents the lower limit for which committed undo is kept. That is, if this parameter is set to 10 minutes, undo retention will be adjusted upwards of 10 minutes if required by the system workload, but will never (unless under space pressure) be tuned or adjusted to below 10 minutes. Thus, the main purpose of this parameter in Oracle Database 10g Release 1 is to support Flashback features as the read-consistency aspect of undo retention no longer needs manual configuration. However, this parameter does impact the size of the Undo

Best Practice: In Oracle Database 10g Release 1, set UNDO_RETENTION to your business requirements for Flashback capabilities.

tablespace. The bigger the value of this parameter, the more space is needed. To determine the appropriate value of UNDO_RETENTION, it is required that the space/undo retention trade off be known. This information is provided by the Undo Advisor. The Undo Advisor analyzes historical workload undo statistics by calculating the total undo generated by the workload over the analysis period specified (this period is restricted based on the Automatic Workload Repository retention policy), the maximum undo generation rate and produces a table that shows undo retention value for different tablespace sizes and vice versa. Depending on the analysis period specified, the Undo Advisor may look at V\$UNDOSTAT or WRH\$_UNDOSTAT (Automatic Workload Repository view), or both. The figure below shows the Enterprise Manager interface for the Undo Advisor output. The graph of Required Tablespace Size versus Undo Retention time clearly shows how to configure Undo tablespace size and initialization parameter UNDO_RETENTION.

Best Practice: In Oracle Database 10g Release 2, you don't need to set UNDO_RETENTION.

In Oracle Database 10g Release 2, auto-tuning of undo retention has been further enhanced. In this release, undo retention is tuned to the maximum allowable by the tablespace, even if it is greater than the longest running query in the system. This gives the best possible retention and in effect makes the setting of the parameter UNDO_RETENTION unnecessary. Oracle now gives you the best possible undo retention and the only consideration in undo management is now the proper sizing of the Undo tablespace. We discuss the topic next.

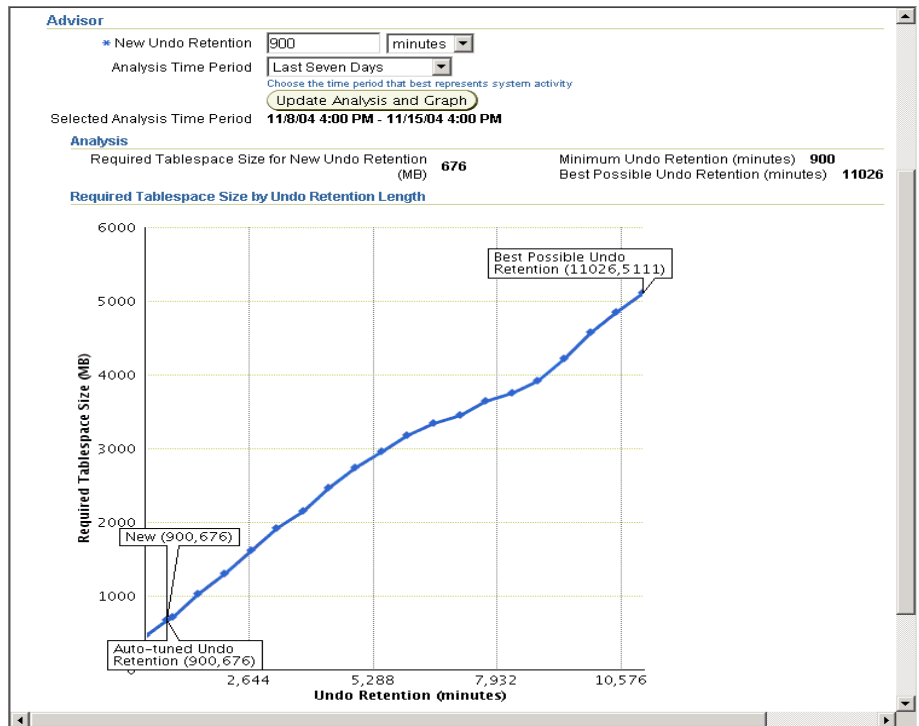


Figure 2: Undo Advisor

Undo Tablespace Configuration

As mentioned earlier, to enable Automatic Undo Management, you must create an Undo tablespace in addition to setting initialization parameter `UNDO_MANAGEMENT=AUTO`. You can create the Undo tablespace as part of `CREATE DATABASE` command or separately as shown below:

```
CREATE UNDO TABLESPACE undotbs DATAFILE  
    'undo.dbf' SIZE 500M REUSE AUTOEXTEND ON;
```

You can create multiple Undo tablespaces in a database but only one of them can be active at any one time. A particular Undo tablespace can be activated by using the `ALTER SYSTEM SET UNDO_TABLESPACE` command. The only exception to one rule of active Undo tablespace per database is in Real Application Cluster (RAC) environments. In RAC environments, each instance is associated with its own Undo tablespace. It should be noted that any instance can read data from Undo tablespaces associated with other instances for read consistency purposes. The instances can also update other undo tablespace during transaction recovery as long as it is not already being used by another instance for undo generation or transaction recovery purposes.

An improperly configured Undo tablespace can easily result in queries failing with the ORA-1555 (snapshot too old) error. If the Undo tablespace is too small, undo of committed transactions, i.e., committed undo, will be rapidly overwritten by new transactions. This greatly increases the chances of encountering ORA-1555 when executing long running queries, as they will inevitably need to go to undo segments for providing read consistency, and if the committed undo data has been overwritten, this error will be raised. Thus, proper sizing of Undo tablespace is essential to avoiding this error.

Initial Setting

The most accurate (and recommended) way of sizing the Undo tablespace is by using the Undo Advisor. The Undo Advisor, which was introduced in Oracle Database 10g to facilitate the undo management, analyzes historical undo statistics and gives recommendations on undo settings including the size of the Undo tablespace. However, a newly created database does not have the kind of historical workload information that is needed by the advisor to make sound recommendations. For such new systems, the best way to size the Undo tablespace is to start with a small size, say 500 MB, but to set the `AUTOEXTEND` datafile attribute to `ON`. This will allow the tablespace to grow automatically as needed. The growth of the tablespace is influenced by 2 factors. First factor is the amount of undo generated. The Undo tablespace will continue to grow to make sure that enough space is available to store *active* undo. Second, the tablespace will grow to try and maintain undo retention ahead of the longest running query. This means that if the longest running query takes 20 minutes, Oracle will try to keep undo information for at least 20 minutes before it is marked as expired and released. If maintaining undo to keep ahead of the longest running query means that the tablespace has to grow, then it will do so as long as the `AUTOEXTEND` datafile

Best Practice: For a new system with insufficient historical undo information, create small tablespace but enable `AUTOEXTEND` datafile attribute.

attribute is enabled. For initial configuration of Undo tablespace, this is our recommended practice.

Final Setting

After the database has been running reached steady state, it is recommended to fix the size of the tablespace. This is because you don't want one runaway transaction to take up all available space and cause the entire system to come to a halt. To determine the appropriate size of the Undo tablespace, you should use the Undo Advisor. Using the maximum allowable time for the "Analysis Time Period" field and specifying the required length for Flashback purposes in the "New Undo Retention" field, run the Undo Advisor to determine the required Undo tablespace size. Add 20% to this size for safety purposes, and disable the AUTOEXTEND attribute and set the Undo tablespace accordingly.

Best Practice: For a system that has reached steady state, fix size of Undo tablespace using Undo Advisor and adding a safety margin of 20%.

PROACTIVE ERROR PREVENTION AND AUTOMATIC PROBLEM HANDLING

We have so far discussed general space and undo management practices that all DBA's must contend with on a regular basis. Oracle has made significant advances that not only greatly simplify administration of space and undo, but also optimize space utilization, enhance transaction and query performance, and substantially reduce the occurrence of common errors and problems. We will now discuss some more advanced and completely optional practices that are primarily meant to show examples to advanced users how they can take further advantage of the new infrastructure provided in Oracle database 10g and take database self-management to the next level.

Over time system requirements change and require reconfiguration. Even in the best-configured system, errors or exceptions will eventually occur. It is a good practice to design systems that can proactively avoid problems but if problems or exceptions do occur, they should handle them gracefully. Oracle provides the infrastructure to handle such situations gracefully so that impact of space and undo management problems such as out-of-space and snapshot-too-old errors can be minimized. Oracle Database 10g provides the Server Generated Alerts infrastructure for alerting to some of the most common types of exceptions. This includes space alerts when a tablespace is running out of space, Long Query Alert when ORA-1555 occurs, alerts associated with space problems in the Flash Recovery Area, and a host of other performance metric alerts. You can easily design a system that automatically prevents, detects and handles these problems, and at the same time lets the DBA know what has happened so that he/she can later investigate the root-cause of the problem and take further action if necessary.

To do this we need to utilize the Server Generated Alert infrastructure and Oracle's Advanced Queuing (AQ) callback capability. This is possible because Oracle 10g's Server Generated Alerts use the AQ infrastructure. A new queue called *ALERT_QUE* is created by default in all Oracle 10g databases, and is used for all alerts generated. A user can subscribe to this queue to catch a particular exception, e.g., ORA-1555 (Long Query Alert) or tablespace full condition, and then register a user-defined callback procedure (this could be a PL/SQL, JMS, or OCI function), so that every time the particular exception occurs the callback procedure is triggered. This callback procedure will contain the user-defined remedy for the exception. For example, for an ORA-1555 exception, the callback procedure remedy could call the Undo Advisor to determine the right size of the Undo tablespace, and then adjust its size accordingly. As a result, the next time the user runs the same transaction or query that resulted in ORA-1555, the error will not occur because the Undo tablespace would have been adjusted automatically. The example below shows in a step-by-step fashion how AQ's can be used for addressing the different types of problems.

1. Create two new subscribers, UNDO_SUB and SPACE_SUB, to the alert queue, ALERT_QUE, in order to catch ORA-1555 and tablespace out of

space condition. Refer to Examples 1 and 2 in the appendix for a sample code on agent subscription.

2. Next create a callback procedure that contains the remedy for the exceptions. For example, the remedy could involve running Segment Advisor and implementing its recommendations when a particular tablespace is running out of space. Sample code for a PL/SQL callback procedure for subscriber SPACE_SUB is shown in Example 3 in the appendix.
3. The third and final step is to register the callback procedure. The sample code in Example 4 in the appendix shows how the procedure myCALLBACK is registered to subscriber SPACE_SUB.

Now every time an alert is generated for which the user has subscribed, depending on the type of alert a corresponding remedy will be applied immediately via the callback procedure.

Next, we will discuss in detail the actual remedies for commonly encountered exceptions in Oracle10g.

1. Space related exception handling

The most common space problems are with tablespaces running out of space. In Oracle9i, the Resumable Space Allocation feature was introduced. This feature prevents space related problems from aborting transactions by instructing the database to suspend any operation that encounters an out-of-space condition. This provides an opportunity for corrective action to be taken while the operation is suspended and *automatically* resume its execution once the problem has been resolved. As discussed earlier, in Oracle Database 10g a warning alert is generated when a tablespace is 85% full and a critical alert is generated when it is 97% full (these are the default values for the thresholds and can be modified by the user). Thus, anytime a session marked as resumable (this is done by issuing the ALTER SESSION ENABLE RESUMABLE command in the session) encounters an out of space condition, the session will go into a suspend mode, while a space alert is automatically generated. The alert in turn triggers the callback procedure associated with the alert, which contains the remedy. For out of space conditions, the remedy can include one or more of the following actions:

- Use Online Segment Shrink to recover wasted space from the tablespace in question. This could fix the problem without requiring addition of new datafile. The following SQL command shrinks the segment specified:

```
ALTER TABLE <table_name> SHRINK SPACE;
```

A more sophisticated way of handling this problem is by using the Segment Advisor. You can create a SQL*Plus script that calls the Segment Advisor using its PL/SQL interface and implements its recommendations automatically. Refer to Example 5 in the appendix for to see how this can be done.

- Add new datafile to the tablespace. By using the Oracle Managed Files (OMF) feature, this script can be made fairly generic without having to hard code the actual name, location or size of the datafile.

2. ORA-1555 “Snapshot too old” error handling

Oracle Database 10g generates the Long Query Alert when ORA-1555 is encountered. For this alert, the corrective action would involve increasing size of the Undo tablespace. For more sophisticated handling, the Undo Advisor can be called using PL/SQL package, DBMS_UNDO_ADV, and its output used to determine the exact amount of space needed. Refer to Example 6 in the appendix to see how the adjustment of the Undo tablespace size based on the Undo Advisor recommendation can be automated. This would correct the exception condition and the transaction or query can then be resubmitted for successful completion, thus obviating the need for external intervention.

Oracle Database 10g provides the basic infrastructure for automatic handling of many of the most common types of problems with minimal effort. Many aspects of database management have been automated in the 10g release of Oracle database, and the way to automate even more has just been described. Users now have the option to design systems that can automatically handle many of the most common types of errors and problems

CONCLUSION

For database administrators, space and undo management continues to be an area of great importance. Oracle has introduced several new features over the years that have simplified administration of this very important area, significantly changing the way space and undo is managed. By following the best practices outlined above, database administrators will maximize space utilization, prevent unwanted exceptions, and most importantly simplify their job.

ACKNOWLEDGEMENTS

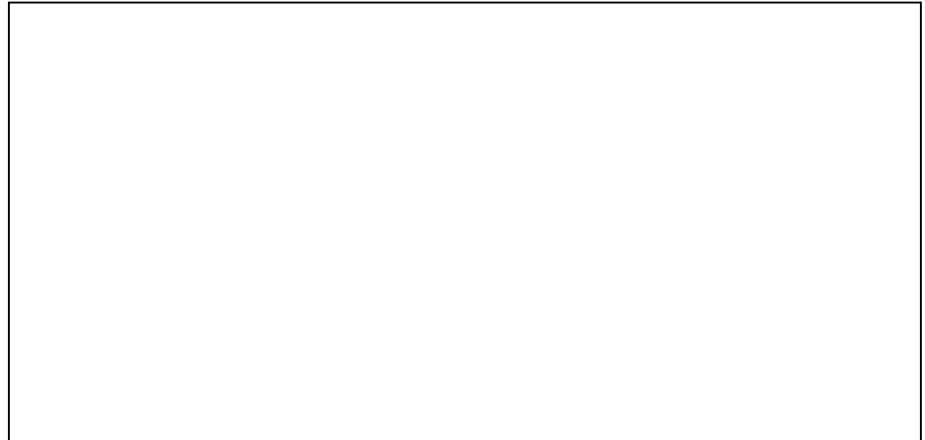
The content of this paper was developed with the help of Cecelia Gervasio, Vasudha Krishnaswamy and Wanli Yang. We thank them for their contribution and feedback.

APPENDIX A: EXAMPLES

Example 1: ORA-1555 Error Subscription

```
DECLARE
  subscriber      aq$_agent;
BEGIN
  subscriber := aq$_agent('UNDO_SUB','ALERT_QUE',null);
  dbms_aqadm.add_subscriber(
    queue_name => 'ALERT_QUE',
    subscriber => subscriber,
    rule       => 'tab.user_data.reason_id = 10 or
                 tab.user_data.reason_id =11' );
END;
/
```

Example 2: Tablespace Full Exception Subscription



Note: The only difference between the 2 subscriptions is the subscriber name and its rule.

Example 3: Sample Code for PL/SQL Callback Procedure

```
create or replace procedure myCALLBACK(
  context raw, reginfo sys.aq$_reg_info, descr
sys.aq$_descriptor,
  payload raw, payloadl number)
AS
  dequeue_options    DBMS_AQ.dequeue_options_t;
  message_properties DBMS_AQ.message_properties_t;
  message_handle     RAW(16);
  message            ALERT_TYPE;
BEGIN
  dequeue_options.consumer_name := 'SPACE_SUB' ;

  -- Dequeue the message
  DBMS_AQ.DEQUEUE(queue_name => 'SYS.ALERT_QUE',
                  dequeue_options => dequeue_options,
                  message_properties=> message_properties,
                  payload => message,
                  msgid => message_handle);

  commit;
-- provide your remedy here, e.g., run Segment Shrink
  commit;
END;
/
```

Example 4: Sample Code for Registering PL/SQL Callback Procedure

```
DECLARE
  reginfo1    sys.aq$_reg_info;
  reginfo1list sys.aq$_reg_info_list;
BEGIN
  reginfo1 := sys.aq$_reg_info('SYS.ALERT_QUE:UNDO_SUB',
                              DBMS_AQ.NAMESPACE_AQ,
                              'plsql://SYS.MYCALLBACK',
                              HEXTORAW('FF'));

  -- Create the registration info list
  reginfo1list := sys.aq$_reg_info_list(reginfo1);

  sys.dbms_aq.register(reginfo1list, 1);
END;
/
```

Example 5: Segment Advisor Sample Code

```
variable id number;
begin
declare
  name varchar2(100) ; descr varchar2(500) ; objid number;
begin
  name := ' ' ;
  descr := 'Segment Advisor Demo';
  dbms_advisor.create_task('Segment Advisor', :id, name,
descr, NULL) ;
  dbms_advisor.create_object(name, 'TABLESPACE','USERS',NULL,
NULL, NULL, objid);
  dbms_advisor.set_task_parameter(name, 'RECOMMEND_ALL',
'TRUE') ;
  dbms_advisor.execute_task(name) ;
end ;
end ;
/

set echo off
spool shrink.sql
select attr1 || ';' from dba_advisor_actions where task_id=:id
;
spool off
@shrink
```

Note: This script runs the Segment Advisor on the tablespace 'USERS' and then shrinks all segments in it that are good candidates for shrinking.

**Example 6: Undo Tablespace Size Adjustment based on Undo Advisor
Recommendation**

```
create or replace procedure increase_utbs_size(newsize_k IN
NUMBER)
is
  utbname          varchar2(30);
  cur_utbsize_k   number;
  add_utbsize_k   number;
  add_file_sql    varchar2(1024);
  cid             number;
  rows           number;
begin
  -- get current undo tablespace name
  select upper(value) into utbname from v$parameter
  where upper(name)='UNDO_TABLESPACE';

  -- get current undo tablespace size
  select round(sum(bytes)/1024) into cur_utbsize_k from
dba_data_files
where upper(tablespace_name)=utbname;
  dbms_output.put_line('Current undo ts ' || utbname || ' is ' ||
                        cur_utbsize_k || 'K');

  -- compute how many more K of datafile to add
  if newsize_k > cur_utbsize_k then
    add_utbsize_k := newsize_k - cur_utbsize_k;
    dbms_output.put_line('Adding ' || add_utbsize_k ||
                        'K datafile to ' || utbname || '...');
  else
    dbms_output.put_line('Undo ts ' || utbname || ' has enough
space.');
```

```
    return;
  end if;

  add_file_sql := 'alter tablespace ' || utbname ||
                  ' add datafile size ' || add_utbsize_k || 'K';

  cid := dbms_sql.open_cursor;
  dbms_sql.parse(cid, add_file_sql, dbms_sql.native);
  rows := dbms_sql.execute(cid);
  dbms_sql.close_cursor(cid);
  dbms_output.put_line('Done. Undo tablespace ' || utbname ||
                        ' is now ' || newsize_k || 'K.');
```

```
end increase_utbs_size;
/

declare
  mql number;
  reqsz number;
  reqsz_k number;
begin
  mql := dbms_undo_adv.required_retention();
  dbms_output.put_line('Required Retention: ' || mql);
  reqsz := dbms_undo_adv.required_undo_size(mql);
  dbms_output.put_line('Required Undo Size: ' || reqsz || 'M');
  reqsz_k := round(reqsz * 1024);

  increase_utbs_size(reqsz_k);
end;
/
```

Note: The sample code above assumes that DB_CREATE_FILE_DEST parameter for Oracle Managed Files (OMF) is set.



White Paper Title: The Do's and Don'ts of Space & Undo Management: Best Practices for Oracle Database 10g
December 2004

Authors: Sujatha Muthulingam, Mughees A. Minhas

Contributing Authors:

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

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
www.oracle.com

Copyright © 2004, Oracle. All rights reserved.

This document is provided for information purposes only
and the contents hereof are subject to change without notice.

This document is not warranted to be error-free, nor subject to
any other warranties or conditions, whether expressed orally
or implied in law, including implied warranties and conditions of
merchantability or fitness for a particular purpose. We specifically
disclaim any liability with respect to this document and no
contractual obligations are formed either directly or indirectly
by this document. This document may not be reproduced or
transmitted in any form or by any means, electronic or mechanical,
for any purpose, without our prior written permission.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective owners.