Space Management

Ari Statham

Oracle DBA

Data4 Limited

Scope

Businesses are becoming more and more dependant on the availability of corporate databases and the business information they hold. To ensure that databases are constantly available and can grow as required, incurring only minimal performance penalties, administration of the database must include space management.

This paper examines why space management is important and the information requirements for a DBA to manage space within an Oracle database. The paper also describes space management techniques including reorganisation and data archiving.

Exclusions

These subjects might be considered to be related to Space Management, but are not in the scope of this paper.

- Oracle Managed Files
- Oracle Flexible Architecture
- Volume / Filesystem Design and Management
- Archive Log Management
- Datafile Mapping / Distribution
- Undo Space Management
- Transportable Tablespaces

Disclaimer

This document does not attempt to replace the need for Oracle manuals, bug databases or experimentation. The aim of the document is to bring attention to the challenges, concepts, features and techniques that DBA's should know about in order to manage the space inside of Oracle databases.

Importance of Space Management

Space management is important in all Oracle databases to some degree. One or more of the following aims will usually be applicable.

Aims of Space Management

Handle Data Growth

The most critical deliverable of good space management is the assurance that all the databases' functions will be performed without failures that are caused by a lack of capacity.

Maintain Data Density

Space management can have a significant influence over the performance of an active database, by maintaining the density of the data stored in the various database structures.

Reduce Administration Effort

The workload of the DBA can be reduced by using sensible space management policies and monitoring software. Various tools and database features can make detecting and correcting capacity problems easier.

Avoiding outages and large reorganisations should also be an aim of space management.

Increase Efficiency of Data Storage / Retrieval

If disk capacity is a restricted resource, then space management policies can be used to control the size of a database.

The way that objects are stored in the database can affect the efficiency of I/O and caching.

Space Management Theory

Provision for the Growth of Data

Most live databases grow over time.

The three challenges that growth presents are

- To ensure the availability of the application by providing adequate capacity
- To maintain acceptable performance levels
- To keep the database manageable in regards to administration tasks

Oracle databases allow space for growth at a number of levels

- Tablespaces have free blocks for new extents
- Extents have free (unused) blocks for new data
- Blocks have free bytes for new (or growing) rows
- Rows may have allocated space for new data (depending on the datatypes and order of the columns)

Pre-allocating space to avoid growing pains can have the side effects of decreased data density, wasted space and increased effort & resource requirements when performing administration tasks, such as backups.

Therefore, if space or performance is an important factor, then some care should be taken when managing space to create an efficient database.

Table and Index Expansion

Number of Extents

In years past, the number of extents that could form a segment was limited by the block size. (The extent map had to fit within half of the segment header block). Since Oracle 7.3, segments have been freed from this restriction by allowing the chaining of additional extent map blocks.

It is amazing to find that there are still databases today with application tables and indexes that have a limited number of extents. Eliminating the risk of transactions failing due to segments being unable to extend is surely important enough to make using the 'unlimited extents' feature universal.

There are still reasons to control the number and size of extents, which are discussed in the next section, "Multiple Extents".

Free Space

The tablespaces that hold growing objects should have enough chunks of free space to accommodate new extents for the near future (at least). Pre-allocating several years worth of free space might not be an option when disk space or backup capacity is limited. A useful alternative is to allow each tablespace to grow dynamically as required.

Multiple Extents

How Many Extents Are Too Many?

Before insisting on a reorg to recreate an object because it has multiple extents, consider if there is any justification for this decision.

A DBA should be sure of the benefits before taking an action that has risks and costs associated to it. Recreating the object in fewer and larger extents may leave a number of small fragments of free space in the tablespace. An outage, backups, planning and DBA time may be required in a large production environment. Having multiple extents can be important to the application's performance in some situations, e.g. when parallel queries rely on a table's extents being physically located on different disks.

Access to the extent map isn't required when a block is referred to directly. This type of access occurs with reads of undo blocks, index blocks, header blocks and table blocks via indexes.

Full table scans or fast full index scans may need to access the extent map, but this information will often be cached, and the cost of accessing it is insignificant when compared to the size of the data being processed.

Dropping an object with thousands of extents can take longer than an object with one extent, (especially if not locally managed), but the performance of dropping an object is not usually a concern. If it is a regular occurrence, then the extents should be sized appropriately (or temporary tables should be utilised).

Dictionary Managed Tablespaces

Free extents are stored in *fet$* and are clustered with tablespace rows. Default sizing allows for 1 block per tablespace, (i.e. approximately 500 free extents per tablespace assuming an 8KB block).

Used extents are stored in *uet$* and are clustered with segment rows. Default sizing allows for an average of 5 extents per segment. (The space in a block is shared, so if some segments have more, others may have less).

If the extent information is larger than will fit into one block, it is put into a new chained block.

The number of extents does not affect parsing, and tests will show no significant effect on segment access. It can be assumed that the data dictionary performance would be adversely affected by segments being allowed to grow thousands of extents. The degree of this effect would be difficult to measure, but because of the various levels of caching involved, it seems unlikely to be significant.

Another thing to consider is contention for the sole ST (Space management Transaction) enqueue. Whenever *uet$* or *fet$* are changed, (i.e. adding or removing extents, or coalescing free extents), the ST enqueue is held, preventing concurrent space management transactions.

Locally Managed Tablespaces

When a segment has more extents than can fit into a segment header block, extent map blocks are created. An 8KB block can hold about 1000 extent entries, (and 500 in the segment header). If the extent size is 1MB, then the overhead of a full table scan is performing a sequential read of a single 8KB block after reading the first 500MB of data, and then again after each subsequent 1000MB.

Selecting from *DBA_EXTENTS* will have to read all segment header and extent map blocks instead of using the data dictionary. This seems unlikely to cause a performance problem.

Extent Size

The extent size is more important than the number of extents. It affects the capacity required for storage and the efficiency of data retrieval.

The smaller the extent size, the less storage space is wasted. Off-the-shelf applications commonly have many small or empty tables and indexes. Using large extents for these objects would waste a lot of space.

When index or table segments are fully scanned, the number of blocks read with one I/O request is the minimum of the extent size and the multiblock read count. If a segment has more than one extent, and that extent is not a multiple of the multiblock read count then its size is inefficient for full scans.

Even though these scattered block reads may not translate to contiguous reads on the disk device, (due to the conspiracy of filesystems, volume management, raid devices, buffering and multi-user environments), the number of I/O requests can be minimized by choosing an extent that is a multiple of the multiblock read count.

"A Reorg Made it Faster"

There are many reasons that recreating a table and its indexes can affect the performance of an application.

When a performance improvement is noticed, it is most likely to be because the data has been repacked into blocks more efficiently; not because the number of extents has been decreased.

Related Segment Expansion

As a database grows, there may be an increased use of

- Temporary segment space
- Rollback / Undo segment space
- Working table space

These structures should not be considered static if the data in the database is growing.

Partitioning

This feature can be extremely useful to create a manageable and scalable database, by partitioning the tables and indexes that hold the growing data.

It is especially good at handling rolling data, e.g. data with recent dates replacing data with old dates.

Data Archiving

As a database grows, more resources are required to perform existing application functions. Although managing the extra demand for archive log storage, latches, I/O bandwidth, network bandwidth, memory and CPU resources are not within the scope of this paper, it should be mentioned that archiving data can be an effective method to control this demand.

Often the most recent data in a database is more frequently accessed than old data. The old data might be summarised and migrated out of the database, or at least, out of the active application tables.

Removing historical data will help maintain the performance levels of the application and administration functions.

Data archiving can be built into the application, implemented using Oracle's partitioning option, or carried out using a third party tool.

Data Density

Space management is concerned with controlling how much space is required to store databases' data. Sparsely stored data requires more disk capacity, CPU power (handling), I/O bandwidth and memory for buffering than densely packed data.

If the performance of a database is important, then the DBA should put some effort into controlling the data density of the relevant objects.

Data density is relevant at many levels of storage, but it really comes down to how many blocks are used to store the data.

Over time, DML and DDL operations can cause data density to change, hence the need to monitor and improve data density.

Reasons Data Could Become Sparse in a Table

Migrated Rows

Migrated [un-chained] rows are stored in two blocks instead of one. It could be considered a sparse row.

This happens when the row grows too large for its current block, and has to migrate the row body to another block. The row header must remain in the original block, because the row will keep the same rowid for its lifetime.

The frequency of row migration depends on

- the settings of *PCTUSED* and *PCTFREE*
- the table's update characteristics
- column additions or modifications
- column order

Columns Dropped or Modified

When rows are made smaller by removing or reducing columns then the blocks will contain more free space. Ideally, this table would be recreated afterwards to 'repack' the rows into the blocks.

A Large Infrequent Delete

A large delete will leave a lot of free space in the blocks under the high water mark. If the delete process is infrequent, then it might be some time before an equivalent level of data density is restored.

Decreasing Clustering Density

If all the data in a table is loaded in the order of an indexed column(s), then the effectiveness of the index is better than if the data was randomly loaded. This is because each key will point to fewer table blocks, reducing the cost of retrieval, (known as the clustering factor).

After a period of inserts and deletes, the clustering factor is likely to decrease. This could be justification of a reorganisation of this table.

PCTUSED and PCTFREE

PCTFREE controls the amount of space in a block reserved for updates to existing rows.

PCTUSED controls when the block is added to the freelist after rows are deleted, i.e. when the block becomes available again for new rows to be inserted.

If a table's rows are not updated at all, or are updated in a fashion that doesn't increase any row's length, then a low *PCTFREE* (e.g. 1%) is appropriate.

If deletes occur in the table, then the *PCTUSED* setting comes into play. The cost of using a high *PCTUSED* setting to increase data density, (the number of rows per block), is the overhead to maintain the freelists. When the sum of *PCTUSED* and *PCTFREE* approaches 100, the blocks are more likely to be placed on and off the freelists with each delete, update or insert.

It is common to see databases where every application table and index has the default settings for *PCTUSED* and *PCTFREE*. In most cases they are reserving more space in each block than is required.

Occasionally, the *PCTFREE* setting is not high enough to prevent row migration in a table.

With 9i's Automatic Segment Management, the configuration of freelists and *PCTUSED* are no longer required. See the section entitled "Space Management Techniques – Prevention of Problems".

Reasons Data Could Become Sparse in an Index

The insertion of random keys naturally results in an average of 75% data density in an index. (Assuming no deletions).

Rebuilding the index will save space, but after DML, it will eventually grow to its natural density again.

The ordered insertion of increasing keys results in an average data density approaching 100%. (Assuming no deletions).

The *PCTFREE* setting is only used when an index is (re)created. A low setting will pack the keys into leaf blocks, but won't allow any room for new keys. Unless the index is a Right Hand Index, (new keys are always higher than the others), or the data is not frequently changed, then the trade off of a low *PCTFREE* setting is that the blocks will have to split to accommodate new keys, taking CPU time and halving the data density for the affected blocks.

Right Hand Indexes

If data is inserted with new keys and deleted data had keys that are never used again then indexes will continue to grow. E.g. key=*sysdate* and rows with old dates are deleted.

While the data in the new leaf blocks will be densely packed, there may be older leaf blocks that are almost empty. Unused branch blocks are never freed, so the index will continue to grow in size. References to unused branch blocks take up space in the branch blocks above, which can increase the number of levels in the index.

Updates to Indexed Values

If the indexed values are updated and the old value ranges aren't used again, then, as above, there will be unused branch blocks not being freed.

A Large Infrequent Delete

In a similar manner to the effect on the density of table blocks, the index blocks can suddenly become sparse after a large delete. The density of the index will remain low until the index is rebuilt or the volume of data deleted is replaced with inserts.

Data Compression

Tables that are not subject to much DML might be suitable candidates for data compression, for example, read only archived data.

Compression ratios can sometimes be increased by ordering the rows in the table, because this may increase the repetition inside each block. If there are more common values in each block, then more data can be replaced with references to the shared symbol table, increasing the data compression ratio.

Index Compression

Repeated keys in index leaf blocks can be replaced with a reference to a shared symbol table. The degree of compression will vary depending on the size and repetition of the keys chosen to be compressed.

Administration Effort

Prevention

If the storage configuration of a database is well planned, then there is often little effort required in actively managing the space in a live database, e.g. using:

- Locally Managed Tablespaces (LMT)
- Automatic Segment Management (ASM)
- Auto-extending datafiles
- Sensible extent sizes
- Flexible rollback segments (using the optimal setting)
- Sensible *PCTFREE/PCTUSED* settings
- Sensible freelist settings

Monitoring and Alerting

Software or scripts can be used to automatically monitor the database and alert the DBA of any potential problems before the queries or transactions fail.

Reorgs

Regular database reorganisations should be avoided. Objects should only be reorganised if there is a real reason to do so. The more recent the version of Oracle, the more ways there are to minimise the loss of availability.

Multiple Extents

If a segment has thousands of extents and its tablespace is managed by the data dictionary, then the performance of dropping the object would be expected to be poor.

This problem can be avoided by using a locally managed tablespace or more appropriate extent sizes in pre-Oracle 8i databases.

Wasted Space

If the size of the database is a concern, then this matter can be addressed at a number of levels.

Row

If there are many rows in a table, then inflated row sizes can have a significant effect, e.g.:

- *CHAR* types instead of *VARCHAR2*
- Null-able columns not trailing
- Number precision and scale too high

Block

The *PCTUSED* and *PCTFREE* settings may be reserving too much free space. (Automatic Segment Management in 9i removes the need to set *PCTUSED*).

Segment

The last extent allocated might be larger than is required to hold the data expected in the near future.

If only direct inserts are used to populate a table, then all new rows are loaded above the high-water mark. If data is also deleted, then the segment will continue to grow, with old blocks being empty or nearly empty.

Tablespace

Wasted space in tablespaces is due to:

- Tablespace fragments that cannot, [or are unlikely to], be used by any segment in that tablespace due to their small size.
- The tablespace having been created larger than is required to hold the data expected in the near future.

Space Management Techniques

Prevention of Problems

Locally Managed Tablespaces

Use LMTs if they are available.

System Allocated Extents

The algorithm behind this is more complex and not well known; therefore its behaviour is less easily predicted. The benefit of using this method of extent allocation is that it will waste little space when there are many small objects and a few large ones in the same tablespace. This is because it will use both small and large extents depending on the size of the object.

Uniform Extent Sizes

Using uniformly sized extents is a simple way to eliminate fragmentation. The only downside to this method of extent allocation is that it may waste space when a tablespace must contain many objects of vastly varying sizes. For example, it is common for off-the-shelf applications to require that many empty tables reside in a certain tablespace that also contains larger, growing tables.

Dictionary Managed Tablespaces

Simulate Locally Managed Uniform Extents

Use a *PCTINCREASE* of 0 and the same value for initial and next extents. If the Oracle version is 8.0, then the minimum extent clause could be used to enforce all extents to be a multiple of a certain value.

Automatic Segment Management

This optional feature of LMTs replaces the use of free lists to manage which into which blocks new data will be inserted. At the beginning of each extent, a number of blocks are reserved for use as bitmap blocks. These bitmaps contain information about each block's status:

- block full
- less than 25% free
- between 25% and 50% free
- between 50% and 75% free
- 75% or more free
- block empty (unformatted)

This feature removes the need for the DBA to manage the free list configuration and *PCTUSED* settings, by providing an automatic method that allows high concurrency. Instead of a block being considered free or not, the capacity range is known for each block, assisting Oracle to utilise space efficiently.

The costs of using ASM are the storage and handling necessary for the bitmap blocks, and that some blocks below the high water mark may be unused, (affecting the efficiency of full table scans and bitmap indexes).

Efficient Extent Sizes

When choosing an extent size, a DBA should consider the expected size of the objects in the tablespace and the multiblock read count. Extents may have to be small if disk capacity is tight, but otherwise they should be a multiple of the multiblock read count that is large enough to prevent a ridiculous number of extents. (It is usually a good idea to set the multiblock read count to the platform maximum).

Data Dictionary Tuning

Some of the data dictionary objects in the *SYSTEM* tablespace may not be sized optimally for a particular application by default.

The file *sql.bsq* is used to make these system objects when the database is created. If a development or test environment has been created previously, then it may be examined to learn how better to allocate space in the production database at creation time.

Segments that would quickly grow to have multiple extents can be created with larger initial and next extent sizes.

Chaining of the cluster blocks can be mitigated by careful adjustment of the *size* value.

In regards to the cluster *c_obj#*, comments in the file *sql.bsq* state:

- A table of 32 columns, 2 indexes (2 columns each) requires a *size* of about 2KB
- A table of 10 columns, 2 indexes, (2 columns each) requires a *size* of about 750B
- The default *size* is 800 Bytes

According to the file *sql.bsq*, the default *size* value for the cluster *c_file#_block#* is based on *5 * sizeof(uet$)*, i.e. 5 used extents per segment.

Careful Use of Direct Inserts

Direct inserts only place new data in blocks that are above the high water mark. Even after deletes, the tables will continue to grow. These tables must be truncated or recreated if the free space is to be recovered.

Tablespace Contents

System Tablespace

The system tablespace should not contain any foreign (application or tool) segments.

It is probably best to leave the system tablespace alone until Oracle 9iR2, when it can be locally managed. The only exception to this might be to employ a minimum extent restriction on the system tablespace to control fragmentation.

Temporary Tablespace

All users should use a dedicated temporary tablespace. In Oracle 9i, a temporary tablespace can be declared as the default, replacing the usual default which is the *SYSTEM* tablespace.

Segments Characteristics

Segments with different characteristics ideally should be placed into different tablespaces. The following factors should be considered:

- Growth of the segment
- Size of the segment
- Type of the segment – rollback, temporary, index, table, partition, etc

Block Sizes

Best Multipurpose Block Size

In most real life (UNIX) situations, direct I/O has been ignored, and the operating system uses some sort of buffering for the datafiles. In these cases, the database block size should match the OS / filesystem buffer block size, because it is more efficient for reading and writing.

Quoting from the Oracle tuning manual:

"Tests have proven that matching the database block size to the UNIX file system (UFS) block size provides the most predictable and efficient performance. "

Benefits of a Larger Block Size

Indexes will have less leaf and branch blocks, resulting in smaller trees. Range scans and index probes may have fewer blocks to access.

Larger rows will be able to fit into a single block, reducing chaining.

Less overhead is required to manage a smaller number of blocks.

Benefits of a Smaller Block Size

With a smaller block size, data is handled at a more granular level, possibly resulting in more efficient buffering and reduced contention.

Non-Standard Choices

Oracle 9i allows tablespaces to have non-standard block sizes. This allows groups of segments to have appropriate block sizes. It also allows the sharing of tablespaces between databases with different block sizes. The administration complexity increases when this option is used, e.g. maintaining a specific buffer pool for the non-standard block size.

Datafile Auto-Extension

If a datafile of each tablespace is set to 'auto-extend' then segments will only fail to find free space for extension if the underlying storage fills up or the datafile reaches its maximum size.

In some environments, e.g. a development database, it can be useful to limit the growth of tablespaces such as the temporary one. Users running ad-hoc statements may make a mistake and cause an auto-extending datafile to grow far larger than is usually necessary. In these cases, it is preferable that the statement fails rather than allow it to continue hogging the system's resources.

Resumable Space Allocation

Instead of failing, rolling back, and reporting an error to the application, the resumable space allocation feature can be activated for a session to suspend a statement that cannot complete due to an inability to allocate extra space. The idea is that the DBA is given a chance to fix the space problem, before the session times out and returns an error message.

This feature may be employed as a safety net, although it really shouldn't be needed if the database's capacity is well configured and monitored.

Detection of Problems

Free Space

Check that sufficient free space for short term expansion exists or can be automatically made available. This applies to temporary, undo, table and index objects.

Number of Extents

Check that no objects are nearing their maximum number of extents.

Tablespace Fragmentation

Check the number and sizes of free space chunks. It may be necessary to examine the extent layout in the datafiles to gain a full understanding of any fragmentation.

Check the sizes of next extents and the value of *PCTINCREASE* for segments in tablespaces that are not locally managed.

Sparse Tables and Indexes

The relevant data in *DBA_INDEXES* and *DBA_TABLES* is updated using the *ANALYZE* command.

For tables, compare the average number of rows per used block to the maximum number of rows that could fit into a block.

For indexes, compare the average number of keys (and *rowids*) that the leaf blocks are storing, compared to the maximum number that could fit into a block.

Further information can be obtained about indexes by viewing the *INDEX_STATS* view or by dumping the index to a trace file.

The supplied package *DBMS_SPACE* can also be used to obtain information about segment block usage.

Chained Rows

Check the *V$SYSSTAT* view for table fetches by continued row, and the *chain_cnt* column of *DBA_TABLES*.

Appropriate Tablespace Usage

Check that

- no application segments are in the *SYSTEM* tablespace
- every user points to the correct temporary tablespace
- all segments are in their correct tablespaces

Clustering Factor

Check the *DBA_INDEXES* view for the number of rows and leaf blocks compared to the *clustering_factor*. A table that is well ordered for a particular index will show a clustering factor that is close to the number of leaf blocks.

Correction of Problems

Index Rebuilds

Indexes can be rebuilt to correct the density, structure or storage parameters.

This can be done online depending on the index type and the Oracle version.

Indexes can also have their leaf blocks coalesced to improve their density, without changing the branch structure.

Table Rebuilding

Tables can be rebuilt to correct the density, structure or storage parameters using the *ALTER TABLE MOVE* command.

This can be done online depending on the table type and the Oracle version.

After rebuilding a table, its indexes require immediate rebuilding because they will have been rendered unusable.

CTAS

Tables can be recreated with rows in a particular order, different storage parameters or with differing data (e.g. simulating a large deletion), using the *CREATE TABLE AS SELECT* and *RENAME* commands. This process can optionally take advantage of parallel processing and nologging parameters.

This may be preferable to exporting and importing or cleaning up after large deletions.

Export / Import

Exporting and importing objects has been a method of reorganisation for a long time. It is possible to customise the DDL, and so alter the storage parameters, as well as repacking the data.

Adjust Storage Parameters

Some parameters can be changed easily, but will obviously not take affect retrospectively. For example, increasing the value for *PCTUSED* will not immediately increase the number of blocks on the free list, and altering *PCTINCREASE* value will not change the sizes of the existing extents. However, these changes will prevent existing problems from becoming worse, without the need for object recreation.

Online Redefinition

This feature uses temporary snapshots as a mechanism to allow a table's attributes to be changed whilst users are running DML statements against it, with minimal locking.

Unfortunately the indexes, triggers and constraints will not have their original names after this process. In Oracle 9iR2, the indexes and constraints can be renamed afterwards.

De-Allocating Unused Space

In the unlikely circumstances that a segment may have had too many extents explicitly allocated, or a non-zero value for *PCTINCREASE* that resulted in the last extent being too large, then the *ALTER TABLE table-name DEALLOCATE UNUSED* command can be used to correct this.

LMTs should be used whenever possible, and extent sizes strictly controlled, so this command should be almost redundant. It is mentioned for completeness.

Feature History

The features mentioned in this paper were introduced with different versions of Oracle.

Oracle 7.3

- Unlimited extents

Oracle 8

- Partitioning

Oracle 8i

- Locally Managed Tablespaces (LMT)
- Index Compression
- Online Index Rebuilding
- Index Coalescing

Oracle 9i

- Automatic Segment Management (ASM)
- Online Index Coalescing
- Online table rebuilding

Oracle 9iR2

- System TS locally managed
- Data Compression

References

- http://www.ixora.com.au
- http://www.oracle-base.com
- http://metalink.oracle.com
- http://asktom.oracle.com
- http://otn.oracle.com
- Myths About Extents
  Cary Millsap/Hotsos LLC