



The Extent of Extents

Why AUTOALLOCATE should be used rather than
UNIFORM extent sizing

An article by of RDB Consulting
January 2008

The Extent of Extents

Why AUTOALLOCATE should be used rather than UNIFORM
extent sizing

To understand the terminology of database storage management we need to define what tablespaces, segments, extents, and space fragmentation is.

What is a tablespace?

A tablespace is a logically continuous area of space that has been allocated to the database in which tables, indexes and other segment types can be stored.

What is a segment?

A segment is a container for database objects that can have data or some other kind of content assigned to it. A segment therefore requires space in a tablespace in which to store its content. There are about 11 segment types in Oracle, but typically a

segment would be a **table, index, lob, rollback** or **temporary** segment. The reference to segments in this paper will primarily mean table and index segment types.

What are extents?

An extent is basically the database term given to the physical chunks of storage space assigned to segments. When you create a segment it has to have at least one physical chunk of space allocated to it which is known as the **initial extent**. A segment cannot exist unless it has at least an initial extent, even if it is completely empty. Once a segment has filled up this chunk of space that it has been allocated, it obviously requires further space in order to continue storing data. The subsequent chunks of space allocated to a segment are known as **next extents** and a segment can have many subsequent next extents. The sum of the sizes of all extents of a segment gives the total size of the storage space allocated to this segment. Extents can also be de-allocated from a segment and be made available as free space within the tablespace. This free space can be reused by any of the other segments residing in that tablespace.

What is space fragmentation?

Space fragmentation within a tablespace happens when there is a lot of allocation and de-allocation of variable sized segments extents. As segments grow and acquire additional extents, these extents get interleaved with all the other segments extents in the same tablespace. When one of the segments gets dropped, shrunk down in size, or reorganized into a single bigger extent, then the pockets of space it previously occupied get freed up and made available for reuse. In order for a pocket of free space to be re-used, the next extent size of the segment requiring the additional space must be the same size or smaller than the pocket of free space. If it is the same size then there is 100% reuse of space and no fragmentation. If it is smaller then we see the single pocket of space being split up into two pockets of space, one getting allocated as an extent and the remaining as free space. So effectively, further fragmentation has now occurred as a single contiguous pocket of space has been split in two. If there is not a big enough pocket of reusable free space for the extent to fit in, then another extent is created out of previously unused free space.

Oracle's MASSIVE Achilles Heel in version 8 and below

In Oracle versions up to and inclusive of version 8, used and free extents were managed in the form of rows in the data dictionary in tables **UET\$** and **FET\$** respectively. A single **ST** (Space Transaction) enqueue was available for all space management tasks resulting in the **serialization** of the allocation and de-allocation of extents.

Another problem with dictionary managed tablespaces was that, in a typical 8K block size database, a maximum number of roughly **500 extents** could be allocated to any particular tablespace without causing **row chaining**. Any extent count over and above this caused row chaining of the dictionary tables, which severely impacted space management performance. It was not uncommon for a segment drop or

reorganization to take hours to complete when it has an excessive amount of extents, and due to the serialization of the space management task, all other allocations or de-allocations would have to wait their turn on the ST enqueue, which could effectively *'hang'* a database.

What did DBA's do to overcome this problem in version 8 and below?

1. Allowed the next extent of a segment to increase its size over the previous extent by using a non-zero PCTINCREASE value, to cater for table growth and prevent the development of excessive extents.
2. Frequently reorganized segments into a single or fewer extents when they had grown too large, preventing the development of excessive extents.
3. Frequently coalesced tablespace free extents to reduce the 500 extent chaining threshold on the dictionary tables, preserving performance.
4. Allowed automated tablespace free space coalescing to take place via the SMON process by setting the tablespace default PCTINCREASE value to a non-zero value.
5. Used the UNIFORM extent size tablespace storage feature of Oracle 8 and above to prevent variable extent sizes and the resulting free space fragmentation. UNIFORM extent sizes meant there was never any wasted free space as a segment's extent would always exactly match any free space pocket size. However, this did not eliminate the dictionary managed space problem and often contributed to it by preventing dynamic segment extent growth.

What was the result of the DBA's course of action in version 8 and below?

1. A non-zero PCTINCREASE value prevents massive extents but creates many variable extent sizes which caused free space fragmentation and the resulting free space wastage.
2. Frequent segment reorganization prevented massive extents but also resulted in a big free space fragmentation problem due to the interleaving of segment extents and the variable size of the extents.
3. Automated tablespace free space coalescing contributed to the free space fragmentation by the fact that the PCTINCREASE value had to be set to a non-zero value and therefore caused variable extent sizes which left variable sized free space pockets when the segment was reorganized and dropped.
4. The UNIFORM extent size feature either wasted a lot of space if the extent size was too large, or caused serious dictionary performance problems if the extent size was too small and the table grew too big.

What did Oracle do to overcome these serious database problems in version 9 and above?

1. Introduced Locally Managed tablespaces where the free and used extents are managed via bitmaps located in the header blocks of the tablespaces datafiles and segments respectively. This totally eliminated the need to store extent and free space information in the dictionary tables and also eliminated the serialized ST space transaction enqueue from space management tasks.
2. Oracle also introduced the AUTOALLOCATE feature of tablespaces where the allocation of extent sizes was done automatically by Oracle thus eliminating the requirement to manually allocate extent sizes to segments or to make use of (often inappropriate) UNIFORM extent sizes.

How does Oracle's AUTOALLOCATE extent size management work?

1. Allocates extents in the following sizes only: 64K, 1MB, 8MB, 64MB or 256MB.
2. A segment gets allocated an initial extent of 64K. As the segment grows in size the sizes of the subsequent extents also increases.
3. The extent allocations make use of the following basic formula, the first 16 extents are 16K in size, the next 64 extents are 1M in size, the next 120 extents are 8M in size, and then they increase to 64M in size and so on.
4. The above sizes are all exact multiples of each other and are also perfectly divisible by 8K, 16K or 32K database block sizes.
5. Although there are multiple extent sizes, fragmentation won't occur as other segments will always be able to find an exact free space size to fit into, or even if free space pockets are split during a space allocation, the remaining size will still always be a multiple of the possible extent sizes.

Benefits of AUTOALLOCATE

1. Extents are always a multiple of the database block size.
2. All extent sizes are multiples of each other.
3. Prevents space fragmentation.
4. Accommodates small segments through to extremely large segments with the same ease.

5. No requirement to group segments into tablespaces according to size.
6. Does not waste space due to an inappropriate size.
7. Won't cause segments to extent into thousands of extents.
8. It is the default for tablespace creation.
9. Requires less work from DBA's when creating tablespaces.
10. Eliminates sizing calculations and DBA sizing miscalculations.
11. Performance benefit with Full Table Scans as extents are fewer and larger and are multiples of the multiblock read count, reducing additional multiblock reads.

Benefits of UNIFORM extent sizes

1. Prevents fragmentation.

Disadvantages of UNIFORM extent sizes.

1. Difficult to calculate appropriate UNIFORM extent sizes that cater for small through to very large segments in databases.
2. Need to group segments together into tablespaces according to size.
3. Miscalculated UNIFORM extent sizes either results in space wastage or forces segments to occupy thousands of extents.
4. Segments frequently out grow their UNIFORM extent size.
5. Performance impact with Full Table Scans if extents are not a multiple of the multiblock read count, necessitating additional multiblock reads in order to retrieve the last remaining blocks of the extent.
6. Tablespace QUOTAS are still dictionary managed and therefore forcing a segment into thousands of extents can still have an impact when reorganizing or dropping it.

Conclusion

1. The advantages of AUTOALLOCATE include and outweigh the advantages of UNIFORM extent size allocations.
2. Miscalculated UNIFORM extent sizes pose a risk to the database.
3. If segments are to be grouped into tablespaces according to size, then it is

strongly suggested not to use multiple UNIFORM extent sizes to differentiate the segments, but rather use multiple BLOCK sizes to differentiate the segments i.e.

3.1 8K tablespace block size for small and medium segments

3.2 16K tablespace block size for large segments

3.3 32K tablespace block size for very large segments.

4. Multiple BLOCK sizes will gave huge performance benefits if used correctly in conjunction with appropriate segment sizing.