ORACLE9I DATABASE ADMINISTRATION BEST PRACTICES

Mughees A. Minhas, Server Technologies, Oracle Corporation.

INTRODUCTION

Oracle has always had the richest functionality of any database. Every new release of Oracle is awash in new features and functionality. Whereas this, for the most part, bodes well for users as each new feature helps make the database system more robust, powerful and efficient, the prolific growth of database functionality also has a direct impact on the administration practices of database administrators. In order to take advantage of the new functionality, database administrators must revisit their current practices with every new release of the software. This paper helps alleviate some of this burden by outlining a number of practices that aptly utilize Oracle9*i* features, thereby enhancing the reliability, availability, manageability, and performance of Oracle9*i* database systems. We do recognize that database environments vary from customer to customer and that best practices have to be formulated in light of the each customer's unique environment. It is therefore, impossible to have an exhaustive discussion of database administration practices in any one paper. In this paper we will focus on administration practices that are generic, that is, independent of platform and environment, but at the same time also vital to good database management. The selected areas can be divided into four categories, i) System and Database Configuration, ii) Space and Object Management, iii) Performance Tuning, and iv) Backup and Recovery. A detailed discussion of each of these categories follows.

SYSTEM & DATABASE CONFIGURATION

A database administrator has to take certain decisions about system and database configuration even before the creation of the database. These decisions are often the most critical and generally very difficult, if not impossible, to undo. If taken correctly they can help avoid significant problems and inconveniences down the road. Determining the optimal configuration of the storage subsystem and the right block size for the database fall under this category. This section looks at issues involved in these areas outlining the best database configuration and creation practices.

STORAGE CONFIGURATION

Configuring the storage subsystem optimally for the database is a very important task for most system and database administrators. A poorly configured storage subsystem can result in I/O bottlenecks that can slow down the fastest machine to a crawl. System administrators, therefore, give due significance to storage configuration and spend great amount of time doing just that. Historically, configuration of storage systems was based on a deep understanding of the database application, which made the configuration process unnecessarily complex and a manageability nightmare. The SAME configuration model offers a better and a more manageable alternative to this. SAME, an acronym for **S**tripe **a**nd **M**irror **E**verything, has been developed by Oracle experts who have done significant work in researching optimal storage configuration for Oracle database systems. This model is based on four simple proposals (i) stripe all files across all disks using a 1 megabyte stripe width, (ii) mirror data for high availability, (iii) subset data by partition and not by disk, and (iv) place frequently accessed data on the outside half of disk drives.

i) *Stripe all files across all disks using a 1 megabyte stripe width*

Striping all files across all disks ensures that full bandwidth of all the disks is available for any operation. This equalizes load across disk drives and eliminates hot spots. Parallel execution and other I/O intensive operations do not get unnecessarily bottlenecked because of disk configuration. Since storage cannot be reconfigured

without a great deal of effort, striping across all disks is the safest, most optimal option. Another benefit of striping across all disks is that it is very easy to manage. Administrators no longer need to move files around to reduce long disk queues, which is a non-trivial administrative task. The easiest way to do such striping is to use volume level striping using volume managers. Volume managers can stripe across hundreds of disks with negligible I/O overhead and are, therefore, the best available option at the present time.

A corollary to this proposal is that storage subsystems should be configured based on required I/O bandwidth and not size of database alone. For example, if the total database size is 100 gigabytes with an I/O bandwidth requirement of 2000 I/O's per second, the system should be configured with 20 disk drives, each 5 gigabytes in size, as opposed to with 2 disk drives of 50 gigabytes each. Both of these configurations can handle 100 gigabytes of data but only one meets the I/O requirement. Storage subsystems should, therefore, be configured based on data volume and I/O bandwidth.

The recommendation of using a stripe size of one megabyte is based on transfer rates and throughputs of modern disks. If the stripe size is very small more time is spent positioning the disk head on the data than in the actual transfer of data. Table 1 below shows the result of our study measuring transfer rates for various stripe sizes. Veritas VxVM volume manager and EMC Symmetrix were used for this study. As the table shows, a stripe size of one megabyte achieves reasonably good throughput and that larger stripe sizes produce only modest improvements. However, given the current trend in advances in disk technology the stripe size may have to be gradually increased.

Table 1: Transfer times for various stripe sizes

I/O Size	Positioning Time	Transfer Time	% Of Time Transferring Data
16 K	10 ms	1 ms	10%
64 K	10 ms	3 ms	23%
256 K	10 ms	12 ms	55%
1 M	**10 ms**	**50 ms**	**83%**
2 M	10 ms	100 ms	91%

For update intensive applications, it is desirable to keep redo logs on separate disks. This is especially true for storage subsystems that cannot cache write operations. Another fact to keep in mind about redo logs is that they are written sequentially. Since many storage subsystems limit the maximum size of single write operation to less than 1 megabyte, a smaller stripe width of 64K for redo logs is more appropriate.

ii) *Mirror data for high availability*

Mirroring data at the storage subsystem level is the best way to avoid data loss. The only way to lose data in a mirrored environment is to lose multiple disks simultaneously. Given that current disk drives are highly reliable, simultaneous multiple disk failure is a very low probability event. Along with disk mirroring, Oracle's multiplexing capabilities should also be utilized. This multiplexing, however, is limited to redo logs and control files only. It is a good practice to multiplex redo logs and control files on top of disk mirroring because multiplexed files are more immune to physical data corruption. In the case of disk mirroring, physical data corruption in a redo log will be reflected on both the mirrors, while in the case of multiplexed files, data corruption in one redo log member will not be duplicated on the other member. Therefore, both redo logs and control files should be mirrored and multiplexed for high availability purposes.

iii) *Subset data by partition and not by disk*

Sometimes it is necessary to separate or partition the overall data set. For instance, a database administrator may want to separate read only data from writeable data on different volumes. In such cases, the partition should be created across all disks, and then concatenated and striped to form a separate volume. This separates the data files logically without compromising any of the advantages of the SAME methodology.

iv) *Place frequently accessed data on the outside half of the disk drives*

The transfer rate of a disk drive varies for different portions of the disk. Outer sectors have a higher transfer rate than inner sectors. Also outer portions of a disk drive can store more data. For this reason, datafiles that are accessed more frequently should be placed on the outside half of the disk. Redo logs and archive logs can undergo significant I/O activity during updates and hence should also be placed on the outer portion of the disks. Since this can lead to an administrative overhead on the part of the database administrator, a simple solution is to leave the inside half of a disk drive empty. The is not as wasteful an option as might appear because due to the circular shape of the disks, typically more than 60% of capacity is on the outside half of the disk. Also the current trends of increasing disk drive capacities make it an even more viable proposition

The primary benefit of the SAME model is that it makes storage configuration very easy and suitable for all workloads. It works equally well for OLTP, data warehouse, and batch workloads. It eliminates I/O hot spots and maximizes bandwidth. Finally, the SAME model is not tied to any storage management solution in the market today and can be implemented with the technology available today. For more details please refer to Oracle technical paper "Optimal Storage Configuration Made Easy" at http://otn.oracle.com/deploy/performance/content.html.

DATABASE BLOCK SIZE

When creating an Oracle database, one of the first decisions that a database administrator has to make involves selecting the database block size. This is done at the database creation time by specifying the initialization parameter DB_BLOCK_SIZE. This parameter can have a value of 2k, 4k, 8k, 16k, or 32k bytes. As a general rule the database block size should be a multiple of operating system block size. A large database block size provides greater efficiency in disk and memory I/O. However, if the block size is too large then each block can contain many logical records and this could lead to block contention in an OLTP environment. Another downside of a very large block size is that for index access where only one row of data is retrieved, more I/O bandwidth is unnecessarily used. Thus, the advantages of a large block should be balanced with its disadvantages. 4k and 8k block sizes are suitable for most environments, where 4k is more appropriate for an OLTP system and 8k for a DSS system.

If an unsuitable block size is chosen at the time of database creation, this situation can be remedied in Oracle9*i*. This involves first creating new tablespaces with the appropriate block size by using the BLOCKSIZE clause. Then using Oracle's online reorganization capability, objects can be moved to the new tablespaces online, resulting in no system downtime. Oracle Enterprise Manager (OEM) has a graphical tool for performing such online reorganizations. It should be noted that online reorganization cannot be performed on the SYSTEM tablespace and hence, the block size of the SYSTEM tablespace is permanent and cannot be changed unless the database is recreated. The best practice, therefore, is to do the due diligence at the database creation time and select the right database block size.

DATABASE CREATION METHOD

The Oracle Database Configuration Assistant (DBCA) is a graphical configuration tool that simplifies the process of creating databases, and altering or removing existing ones. Custom database creation includes specifying various Oracle options such as Spatial, OLAP services, etc., defining sizing parameters, archiving and files locations, storage

attributes, as well as manually running all the standard SQL scripts at the end of a database creation. DBCA performs all the necessary steps for the user with minimal input and eliminates the need to plan in detail the parameters and structure of the database. Furthermore, DBCA also makes recommendations about certain database settings based on user input and thus helps database administrators make correct decisions for optimal database configuration. DBCA can be used to create single instance databases, or create or add instances in an Oracle Real Application Clusters environment.

In Oracle9*i* the concept of database templates was introduced in DBCA, which significantly simplifies the management of databases. Database templates are a new mechanism for storing database definition in XML format. The definition includes all characteristics of the database, such as, initialization parameters, redo log settings, etc., and can be used to create identical databases on local or remote machines. There are two kinds of templates, those that contain only the structure of a database, and those that contain the structure as well as the data. This functionality allows administrators to create new databases using a pre-defined template or create a new template, for later use, based on an existing database. As a result, existing databases can be cloned with or without the data, providing a powerful new way of duplicating databases. Templates can be saved and used as needed to deploy new databases.

For embedded applications deployment or unattended database creation, DBCA can be run in silent mode using a predefined template. This means that database creation tasks can now be scheduled using standard operating system schedulers without requiring database administrator supervision. All these features make DBCA a very rich and powerful tool and we recommend that it be used for setup and creation of all Oracle databases and instances.

SPACE & OBJECT MANAGEMENT

A number of additions have been made in Oracle9*i* to facilitate management of various database objects as well as to enhance their performance. Karl Buttner, President of 170 Systems, applauding the new management capabilities of Oracle9*i* said, *"enhanced administration and database management capabilities allow our customers to reduce their administration and operational costs, while improving availability and performance of their systems"*. Among the most significant management enhancements in Oracle9*i* is the introduction of Automatic Undo Management, a feature that eliminates the need to manage rollback segments manually. Furthermore, improvements have been made in tablespace and segment management that, if used appropriately, reduce the database administrator's workload significantly.

AUTOMATIC UNDO MANAGEMENT

In order to simplify management of rollback segments, Oracle9*i* introduced Automatic Undo Management which enables the database server to automatically manage allocation and management of Undo (Rollback) space among various active sessions. Administrators no longer need to manually create rollback segments. Instead they can simply create an UNDO tablespace and ensure that adequate disk space is allocated to the tablespace. There are many benefits of Automatic Undo Management. The first benefit is that of manageability. Database administrators no longer need to assign certain transactions to specific rollback segments or worry about the size and number of rollback segments. Another benefit is that Automatic Undo Management essentially eliminates the possibility of undo block and consistent read contention. Using this feature, Oracle dynamically 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 no intervention from the administrator. All a database administrator has to do is set initialization parameter UNDO_MANAGEMEN to AUTO, UNDO_RETENTION to the duration of the longest running query, and allocate sufficient space for the Undo tablespace. OEM offers a tool to determine the size of Undo tablespace based on retention time. As an example, to configure a database to support queries that run for 30 minutes or less, the

database administrator can simply set the undo retention parameter to 30 minutes. With retention control, users can configure their systems to allow long running queries to execute successfully without encountering the dreaded ORA-1555 (snapshot too old) error.

LOCALLY MANAGED TABLESPACES

Locally managed tablespaces perform better than dictionary managed tablespaces, are easier to manage, and eliminate space fragmentation related problems. For operations like space allocation and deallocation, they use bitmaps stored in datafile headers and unlike dictionary managed tablespaces do not contend for centrally managed resources. This eliminates the need for many recursive operations that are sometimes required for dictionary managed space allocation such as allocating a new extent. Locally managed tablespaces offer two options for extent management, Auto Allocate and Uniform. In the Auto Allocate option, Oracle determines the size of each extent allocated without database administrator intervention, whereas in Uniform option, the database administrator specifies a size and all extents allocated in that tablespaces are of that size. We recommend using the Auto Allocate option because even though it might result in objects having multiple extents, the user need not worry about it since locally managed tablespaces can handle a large number of extents (over 1000 per object) without any noticeable performance impact. We therefore suggest that database administrators not worry about extent sizes and use the Auto Allocate option.

In Oracle9*i* users can migrate a dictionary managed tablespace to a locally managed one by called the PL/SQL procedure DBMS_SPACE_ADMINISTRATION.TABLESPACE_MIGRATE_TO_LOCAL. This allows databases migrated from prior releases of Oracle to enjoy the benefits of locally managed tablespaces. Starting with the second release of Oracle9*i*, all tablespaces with the exception of SYSTEM tablespace will be created as locally managed tablespaces by default.

TEMPORARY TABLESPACES

Oracle creates temporary segments for different kinds of sort operations. Using permanent tablespaces for such segments unnecessarily incurs the same space management overhead as that of managing permanent data. The right practice is to use a specific kind of tablespace called Temporary Tablespace for this purpose. In Temporary Tablespace space allocation is done by selecting from a pool of extents in the SGA thereby improving concurrency of multiple sort operations, reducing overhead, and avoiding many space management operations altogether. Within a Temporary Tablespace, all sort operations for a given tablespace share the same sort segment. This is a more efficient way of handling temporary segments both from a resource usage and database performance perspective.

Temporary Tablespaces can be locally managed or dictionary managed. Having already discussed the benefits of locally managed tablespaces, the preferred practice is to create a locally managed Temporary Tablespace. Locally managed Temporary Tablespaces do not require backups and can be used with read-only/standby databases. It should be noted that the command for creating this tablespace is different from that of other tablespaces. It is as follows: CREATE TEMPORARY TABLESPACE <name> TEMPFILE <file-name> SIZE <file-size> EXTENT MANAGEMENT LOCAL.

In Oracle9*i* database administrators can specify a *Default Temporary Tablespace* for the entire database. This can be done using the DEFAULT TEMPORARY TABLESPACE clause of the CREATE DATABASE or ALTER DATABASE statement. Using the *Default Temporary Tablespace* option ensures that all users have a temporary tablespace assigned to them. This guarantees that all disk sorting occurs in the temporary tablespace and that SYSTEM tablespace is not mistakenly used for sorting. Additionally, since permanent objects cannot be created in the temporary tablespace, the *Default Temporary Tablespace* feature guarantees that permanent and temporary objects do not share the same tablespace, which is another very good practice.

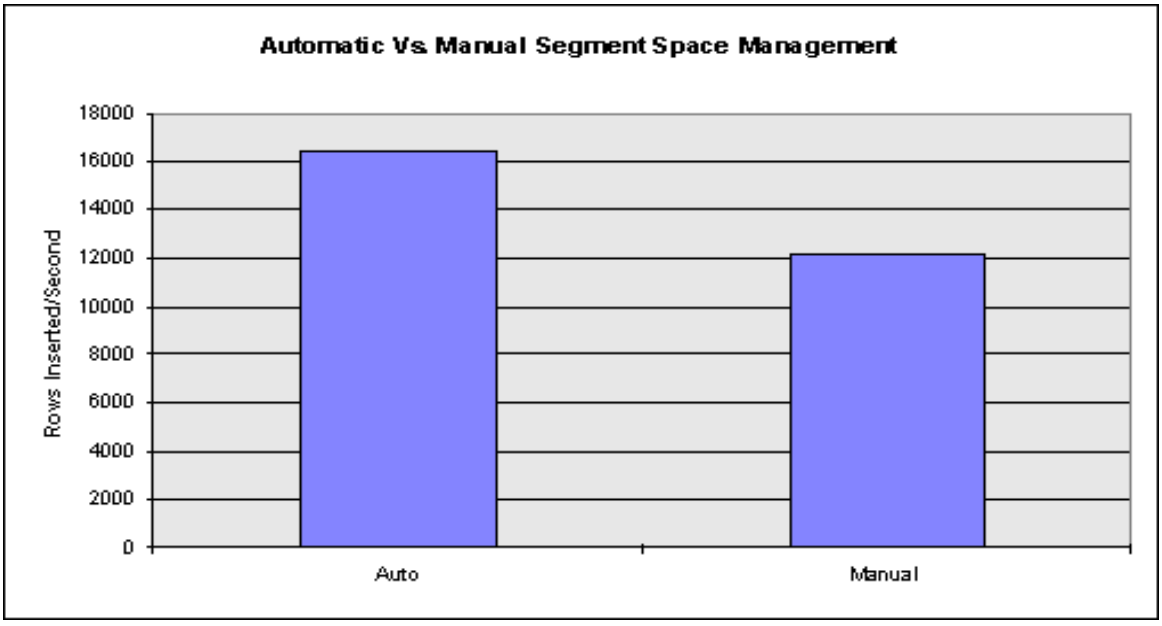
SEGMENT MANAGEMENT

Another issue of concern for database administrators is space management within an object. Starting with Oracle9*i*, database administrators have the option of managing space within an object automatically using the Automatic Segment Space Management feature. It simplifies space administration tasks and eliminates much of performance tuning related to space management. It facilitates management of free space within tables and indexes, improves space utilization, and provides significantly better out-of-box performance and scalability.

Prior to Oracle9*i* administration of space management within a segment was done using data structures called FREELISTS. FREELISTS kept track of blocks within an object with enough free space to allow insertion of new rows. Database administrators defined the number of FREELISTS and FREELIST GROUPS when creating an object, and the table level parameter PCTUSED was used to control how blocks were placed on and removed from a FREELIST. The new mechanism introduced in Oracle9*i* makes space management within an object completely transparent by using bitmaps to track the space utilization of each data block allocated to the object. The state of the bitmap indicates how much free space exists in a given data block (i.e., > 75%, between 50 and 75%, between 25 to 50% or < 25%) as well as whether it its formatted or not. The new implementation relieves database administrators from manually managing space within an object. Another benefit of the Automatic Segment Space Management is that it improves space utilization within a data block. This is because the bitmaps are much better equipped to track and manage free space at the data block level than FREELISTS. This enables better reuse of the available free space particularly for objects with rows of varying size. Additionally, the Automatic Segment Space Management feature improves the performance of concurrent DML operations significantly since different parts of the bitmap can be used simultaneously, eliminating serialization for free space lookups.

The performance and manageability gains provided by Automatic Segment Space Management are particularly noticeable in a Real Application Cluster environment. Figure 1 below shows the results of internal tests conducted by Oracle comparing the performance of automatic and manual segment space management. This test was conducted on a two node (6 x 336 MHz CPU, 4 GB RAM per node) Real Application Cluster database by inserting about 3 million rows in a table. The Automatic Segment Space Management provided over 35% performance gain over an optimally tuned segment (8 FREELIST GROUPS, 20 FREELISTS) using the manual mode.

Figure 1: Automatic Segment Space Management Provides Better Performance



The Automatic Segment Space Management feature is available only with locally managed tablespaces. In the CREATE TABLESPACE command a new clause SEGMENT SPACE MANAGEMENT has been introduced with options of AUTO and MANUAL. AUTO uses Automatic Segment Space Management while MANUAL continues to use FREELISTS for managing free space within the objects.

PERFORMANCE TUNING

A poorly tuned system is only a small step away from a system that is unavailable. Tuning a database system to ensure that it is performing within acceptable guidelines is, therefore, a top database administrator priority. Oracle9*i* has introduced new features that facilitate tuning. Mark Shaiman, a senior research analyst at META Group, agrees – *"While Oracle has always included significant functionality in its DBMS product, it generally requires various "tuning" to achieve optimal performance. (Oracle)9i automates many secondary processes, enabling easier management and DBA user friendliness"*. In this section we will discuss a number of good tuning practices in light of the new features in Oracle9*i*.

OVERALL METHODOLOGY

Performance tuning is not a one-shot task but an ongoing process. Effective database tuning requires good methods and tools. A comprehensive approach to performance tuning must involve the following steps:

i) *Gathering full set of operating system, database, and application statistics regularly*

Database administrators should always have a set of statistics taken a particular point in time, which they should use as baseline. Whether they are trying to discover cause for performance degradation or trying to improve performance, comparing current statistics to baseline statistics allows a database administrator to quickly figure out the main area of concern. This is even more important in the event of migrations and upgrades. Statistics should always be collected before and after any scheduled migration or upgrade so that in the event of unexpected behavior, the database administrator can determine, as quickly as possible, the cause of the problem. To this end, Oracle provides the StatsPack utility. StatsPack is a tool for comprehensive diagnostic of performance problems at the database and application level. The tool can be set up to take periodic 'snapshots' of performance data and store them in persistent tables in the database. The user can define the snapshot interval, i.e., the frequency of gathering data, and also the number of statistics collected. For more details on how to install and run StatsPack, please refer to technical papers, "Diagnosing Performance Using StatsPack, Part I and II" at http://otn.oracle.com/deploy/performance/content.html.

ii) *Identifying problem areas*

The main purpose of gathering statistics is to identify performance problem areas in the database. StatsPack generates an easy-to-read report that identifies areas of potential performance problem. The summary page of a typical StatsPack report is shown in Figure 2. The greatest benefit of this report is seen when there is 'baseline' performance data available for the system to compare with current data as this allows the database administrator to see what actually has changed in the system. For example, if the percentage of shared pool being used for single-use statements goes up considerably, this would indicate the possible use of literal values in applications resulting in additional parsing of the SQL, and thus causing system slowdown.

iii) *Determining potential solutions*

Once problem areas have been identified, determining a solution is a relatively straightforward task. For example, assume that in step (ii) we discovered that use of literal values in stored procedures was causing system

Figure 2: First page of a typical StatsPack report highlights key data about the system.

STATSPACK report for

DB Name	DB Id	Instance	Inst Num	Release	OPS	HOST
PROD	4221457278	PROD	1	9.1.0	YES	mailhost

Start Id	End Id	Start Time	End Time	Snap Length (Minutes)
2327	2329	06/24/2001 11:00	06/24/2001 13:00	120.18

Cache Sizes

db_block_buffers:	38000
db_block_size:	8192
log_buffer:	1048576
shared_pool_size:	150M

Load Profile

	Per Second	Per Transaction
Redo size:	15,689.18	3,922.02
Logical reads:	21,549.53	5,387.01
Block changes:	76.07	19.02
Physical reads:	12.53	3.13
Physical writes:	3.64	0.91
User calls:	210.33	52.58
Parses:	8.13	2.03
Hard parses:	0	0
Sorts:	6.18	1.54
Transactions:	4	
Rows per Sort:	11.53	
Pct Blocks changed / Read:	0.35	
Recursive Call Pct:	14.94	
Rollback / transaction Pct:	2.52	

Instance Efficiency Percentages (Target 100%)

Buffer Nowait Ratio:	100
Buffer Hit Ratio:	99.94
Library Hit Ratio:	100
Redo NoWait Ratio:	100
In-memory Sort Ratio:	99.95
Soft Parse Ratio:	99.94
Latch Hit Ratio:	97.86

Top 5 Wait Events

Event	Waits	WaitTime (cs)	% Total Wt Time
db file sequential read	67,254	73,087	44.77
log file sync	28,252	30,550	18.71
log file parallel write	28,310	26,304	16.11
db file parallel write	2,681	9,430	5.78
global cache cr request	32,825	5,586	3.42

bottlenecks. An obvious solution to this would be to replace literal values with bind variables in stored procedures. If the application is very large, like many ERP applications are, this may not be a viable solution. In this case forcing sharing of cursors by setting the initialization parameter CURSOR_SHARING=FORCE would be a better solution. In this fashion we should outline one or more solutions for each problem area identified.

iv) *Implementing solutions*

The golden rule of database tuning is to change one thing at a time and then measure the difference. Let us assume that multiple problem areas were identified in step (ii), and their potential remedies determined in the subsequent step. The recommended practice is to implement each remedy individually and then measure its impact on the system. However, system downtime requirements may prohibit such a practice. In that case, attempt should be made to implement only those set of remedies together that can be validated independently.

v) *Repeating the process*

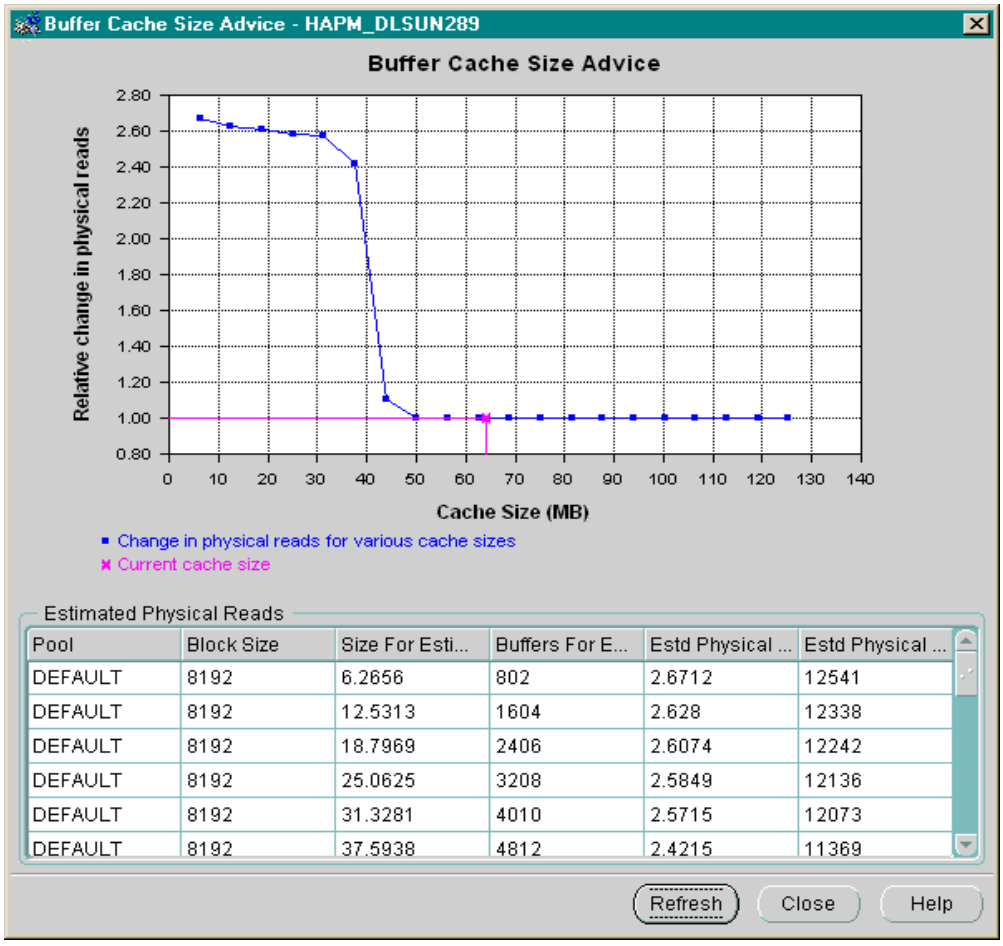
Once a remedy or solution has been implemented, it should be validated. Validation should be done by gauging users perception of performance and also by repeating step (i) and gathering more statistics to see if the remedies

have had the desired impact or not. Additionally, as the workload on the system changes over time, or applications are modified, the whole tuning exercise will have to be repeated.

BUFFER CACHE TUNING

Certain memory structures like the buffer cache are key to the performance of the database. Whereas, the StatsPack had taken mystery out of the tuning process, the importance of a well-tuned buffer cache merits a more rigorous method. To this end the Buffer Cache Advisory was introduced in Oracle9*i*. This advisory relies on an internal simulation based on the current workload to predict the cache "miss" rates for various sizes of the buffer cache raging from 10% to 200% of the current cache size. These predictions are published through a new view V$DB_CACHE_ADVICE. This view can be used to determine the optimal size of the buffer cache for the present workload. OEM provides a handy tool that interprets the data of V$DB_CACHE_ADVICE. The output is illustrated in Figure 3. By providing a deterministic way to size the buffer cache, Oracle9*i* takes the guesswork out of buffer cache configuration.

Figure 3: Oracle9i Buffer Cache Advisory



Database administrators should use this advisory for buffer cache tuning. To turn it on the initialization parameter DB_CACHE_ADVICE must be set to “ON”. There is a minor performance overhead associated with the data collection and cache simulation. However, the benefits of a properly tuned buffer cache far outweigh the overhead. In the second release of Oracle9*i*, advisories for the two remaining memory structures, PGA and Shared Pool, will

also be provided.

SQL EXECUTION MEMORY TUNING

The performance of complex long running queries, typical in a DSS environment, depend to a large extent on the memory available in the Program Global Area (PGA). In Oracle8*i* administrators sized the PGA by carefully adjusting a number of initialization parameters, such as, SORT_AREA_SIZE, HASH_AREA_SIZE, BITMAP_MERGE_AREA_SIZE, and CREATE_BITMAP_AREA_SIZE, etc. Oracle9*i* provides an option to completely automate the management of PGA memory. Administrators merely need to specify the maximum amount of PGA memory available to an instance using a newly introduced initialization parameter PGA_AGGREGATE_TARGET. The database server automatically distributes this memory among various active queries in an intelligent manner so as to ensure maximum performance benefits and the most efficient utilization of memory. Furthermore, Oracle9*i* can adapt itself to changing workload thus utilizing resources efficiently regardless of the load on the system. The amount of the PGA memory available to an instance can be changed dynamically by altering the value of the PGA_AGGREGATE_TARGET parameter making it possible to add to and remove PGA memory from an active instance online. Since the database engine itself is better equipped to determine SQL execution memory requirements, database administrators should use this feature and not try to tune the PGA manually. This should translate to better throughput for large number of users on the system as well as improved response time for queries.

The automatic SQL execution memory management feature is enabled by setting the parameter WORKAREA_SIZE_POLICY to AUTO and by specifying a size of PGA_AGGREGATE_TARGET in the initialization file. These two parameters can also be set dynamically using the ALTER SYSTEM command. In the absence of either of these parameters, the database will revert to manual PGA management mode. In the second release of Oracle9*i* advisory for PGA_AGGREGATE_TARGET will be introduced. Just like in Buffer Cache Advisory, the PGA Advisory will suggest the appropriate size for PGA memory and thus make PGA tuning an even simpler task.

COMMON CAUSES OF PERFORMANCE PROBLEMS

Now that we have discussed best practices in performance tuning, let us look at those practices that cause performance problems. Our recommendation here is simple: “avoid them”.

i) *Bad Connection Management*

The application connects and disconnects for each database interaction. This is a common problem with stateless middle-ware in application servers. This mistake has over two orders of magnitude impact on performance, and it is totally unscalable. A possible solution here is to have a three-tier architecture where users or clients connect to the middle tier that has permanent connections to the database. In this way one permanent connection to the database can be used by different users.

ii) *Bad Cursors Sharing*

Not sharing cursors results in repeated parses. If bind variables are not used, then hard parsing occurs for all SQL statements. This has a significant negative performance impact. Cursors with bind variables that open the cursor and re-execute it many times should be used. For large applications that don’t use bind variables, cursor sharing can be enforced by setting the initialization parameter CURSOR_SHARING=FORCE.

iii) *Getting Database I/O Wrong*

Many sites lay out their databases poorly over the available disks. Other sites specify the number of disks incorrectly, because they configure disks by disk space and not I/O bandwidth. By using the SAME methodology, as discussed earlier in the paper, this problem can be avoided.

iv) *Redo Log Setup Problems*

Many sites run with too few redo logs that are too small. Small redo logs cause frequent log switches which can put a high load on the buffer cache and I/O system. If there are too few redo logs, then the archive process cannot keep up, and the database stalls. As general practice, databases should have at least 3 redo log groups with two members in each group.

v) *Lack of Free Lists, Free List Groups, and Rollback Segments*

Serialization of data blocks in the buffer cache can occur due to lack of free lists, free list groups, transaction slots (INITRANS), or shortage of rollback segments. This is particularly common on INSERT-heavy applications with large database block sizes (8K to 16K). This problem can easily be avoided by using the Automatic Segment Space Management feature along with Automatic Undo Management.

vi) *Long Full Table Scans*

Long full table scans for high volume or interactive online operations could indicate poor transaction design, missing indexes, or poor SQL optimization. Long table scans, by nature, are I/O intensive and do not scale for high number of users. OEM tuning pack offers a powerful tool, SQL Analyze, for identifying and tuning resource intensive SQL. This is a good way of addressing SQL optimization issues.

vii) *On Disk Sorting*

Disk sorts, as opposed to memory sorts, for online operations could indicate poor transaction design, missing indexes, sub-optimal PGA configuration, or poor SQL optimization. On disk sorts, by nature, are I/O intensive and unscalable. By allocating sufficient PGA memory and using Automatic PGA Tuning feature of Oracle9*i* discussed earlier, this problem can be eliminated.

viii) *High Amounts of Recursive SQL*

Large amounts of recursive SQL executed by SYS could indicate space management activities, such as extent allocations, taking place. This is unscalable and impacts user response time. This generally occurs when using dictionary managed tablespaces. Locally managed tablespaces can help improve performance in such situations by significantly reducing recursive SQL.

ix) *Schema Errors and Optimizer*

In many cases, the application uses excessive resources because the schema owning the tables has not been correctly migrated from the development environment or from an older implementation. Examples of this are missing indexes or incorrect statistics. These errors can lead to sub-optimal execution plans and poor interactive user performance. When migrating applications of known performance, export the schema statistics to maintain plan stability using the DBMS_STATS package. Likewise, optimizer parameters set in the initialization parameter file can override proven optimal execution plans. For these reasons schema statistics and optimizer settings should be managed together as a group to ensure consistency of performance.

x) *Use of Nonstandard Initialization Parameters*

These might have been implemented based on poor advice, incorrect assumption or prior need. In particular, parameters associated with spin_count on latches and undocumented optimizer features can cause a great deal of problems that can require considerable investigation.

BACKUP & RECOVERY

To put it simply, bad things can happen. Systems can crash, files can be erroneously deleted, media failures can occur. A database administrator who does not have a sound backup and recovery strategy to deal with such situations must keep his/her resume handy. In this section, we will discuss briefly some procedures and practices that will help database administrators keep their job.

Businesses must have a strategy to recover from disasters that result in data loss. Currently only 45% of Fortune 500 companies have a formal strategy in place to shield themselves from disastrous data loss. Of these only 12% of the strategies are considered effective at the enterprise level[1]. As part of devising a formal strategy, database administrators must plan for data loss due to unexpected outages, such as, hardware failures, software failures, human errors, acts of nature, etc. For each of these possible events, database administrators must have a plan to deal with them. In some cases media recovery may be necessary, while in others instance recovery would suffice. For both of these situations, database administrators must have a plan to deal with it. We will discuss them in detail later in this section. Finally, database and system administrators must document their database environment and recovery strategies so that if a failure does occur, the database administrator is able to diagnose the problem and immediately address it.

INSTANCE RECOVERY PLANNING

Systems can crash due to a number of reasons, e.g., power outage. In this case, instance recovery will take place upon restart of the database before it can become available to the users. The goal of instance recovery is to restore the physical contents of a database to a transactionally consistent state. Recovery, therefore, must apply all changes made by committed transactions that were still in the buffer cache and not written to disk prior to the crash, and rollback all on-disk changes made by the uncommitted transactions. Instance recovery is handled by Oracle internally and does not need any administrator intervention. The only aspect of instance recovery that a database administrator needs to be concerned about is the length of time taken to perform recovery and this can now be easily controlled by setting the parameter FAST_START_MTTR_TARGET in the initialization file. In systems with buffer cache many gigabytes in size, instance recovery can be very long and completely unpredictable. Database administrators have low tolerance for both of these attributes. With the new Fast-Start Time-Based Recovery feature in Oracle9*i*, database administrators can specify the average recovery time (in seconds) from instance crashes using the FAST_START_MTTR_TARGET parameter. This feature is implemented by limiting the number of dirty buffers and the number of redo records generated between the most recent redo record and the last checkpoint. The database server essentially adjusts the checkpoint rate to meet the specified recovery target.

If FAST_START_MTTR_TARGET parameter is set to too low a value, then the database will be writing to disk very frequently and this could easily result in system slowdown. The parameter should therefore be adjusted such that the system is not doing excessive checkpoints but at the same time, the recovery time is still within acceptable limits. The second release of Oracle9*i* will have an advisory called Recovery Cost Estimate for determining the performance

[1] "Calculating the Cost of Downtime", Angela Karr, Advanstar Communications

impact for various settings of the FAST_START_MTTR_TARGET. This feature will greatly simplify the determination of the right value for this parameter.

MEDIA RECOVERY PLANNING

BACKUP STRATEGY

Media recovery is needed anytime there is data loss. The most common causes of data loss are media failures and human errors. In order to enable media recovery, the database must first be placed in ARCHIVELOG mode. This can easily be done using the ALTER DATABASE command. Next, database administrators must devise a backup method. In the past, database administrators wrote their own scripts for backing up the database. This is not advisable anymore since Oracle's Recovery Manager utility is better equipped to manage backups. Recovery Manager has a command line version as well as GUI version that is integrated into OEM. Recovery Manager allows users to specify all their backup requirements, for instance, full online database backup to tape or offline full database backup to disk, etc. Moreover, using OEM's scheduling ability, backup jobs can be set up to run at a specified time on a regular basis or an adhoc basis as needed. It is a good practice to backup the control file along with the database files. At a minimum, the control file should be backed up whenever the database structure changes.

For those administrators who have a need to backup system files as well, Oracle has initiated, in alliance with media management vendors, the Backup Solution Program[2]. Members of this program are third party vendors whose tools provide a comprehensive solution for backing up both system and Oracle database files. This is better than using two separate tools, Recovery Manager or OEM for Oracle files and a separate tool for system files, for managing backups.

As discussed earlier, redo log files should be multiplexed. This means that each redo log will have two or more copies or members. It is highly recommended that at least two members of each redo log group be backed up by the backup job. This underscores the importance of redo logs because if during recovery there is a problem with an archived redo log, either due to physical corruption or human error, then complete recovery cannot be performed on the database, thus resulting in certain data loss. By having a spare backup, the probability of such an event is significantly reduced. Note that database administrators should not backup a copy of redo log and then back up the same copy twice. Instead they should backup the two redo log mirrors separately so that in case one of them is corrupted, a second redo log is available for recovery. If both the copies are of the same archived redo log then we are not safe against log corruption events.

RECOVERY STRATEGY

Database administrators must simulate data loss scenarios and test their recovery plans. Unfortunately, most database administrators do not test their backups or their recovery strategies. Testing out the recovery plan helps validate the processes in place and also keeps the administrators abreast of recovery issues and pitfalls. The four major causes of data loss are media failures, human errors, data corruption, and natural disasters. A recovery plan must be devised to deal with each event.

i) *Media Failure*

A common example of media failure is a disk head crash that causes the loss of all database files on a disk drive. All files associated with a database are vulnerable to a disk crash, including datafiles, control files, online redo logs, and archived logs. Recovery from media failure involves restoring all the affected files from backup and then

[2] http://otn.oracle.com/deploy/availability

performing recovery of them. In the worst-case scenario, the full database may have to be restored and recovered. Media failure is the primary concern of a backup and recovery strategy, because it typically requires restoring some or all database files and the application of redo during recovery.

ii) *Human Error*

Users can accidentally delete data or drop tables. A good way to minimize such situations is to grant only those database privileges to the user that are absolutely required. But human errors will still occur and database administrators must be prepared to deal with them. Performing logical backups or exports of tables that are likely targets of human errors is a good way to handle accidental table drops. Flashback Query, a new feature introduced in Oracle9*i*, is very good at recovering from accidental deletes or inserts. As a final resort, the entire tablespace containing the object affected by the human error may have to be restored from backup and recovered. These are various recovery methods for human errors that should all be tested.

iii) *Data Block Corruption*

Faulty hardware or an operating system bug can result in an Oracle block that is not in a recognizable Oracle format, or whose contents are internally inconsistent. Oracle has the Block Media Recovery feature that restores and recovers corrupt blocks only and does not require for an entire datafile to be recovered. This lowers the Mean Time to Recovery (MTTR) because only blocks needing recovery are restored and recovered while the effected datafiles remain online.

iv) *Disasters*

Earthquakes and floods can result in entire data center facility being damaged. Such events require restore and recovery at an alternate site. This necessitates that backup copies kept at the alternate site in addition to the primary site. Off-site restores should be thoroughly tested and timed to ensure that businesses can survive such disasters without compromising their service level agreements.

By thoroughly testing and documenting the recovery plan database administrators can substantially limit unexpected outages and enhance their database system's availability.

CONCLUSION

Businesses have come to regard their IT infrastructure as a strategic imperative. Given that database management systems are at the core of the IT infrastructure, their reliability and availability are of principal importance to businesses. In this paper we have outlined some key practices in the area of database configuration, performance, management, and recovery, that will significantly improve the reliability and availability of the database, as well as reduce operational and administrations costs to the businesses. With even minimal database downtime becoming unacceptable for an increasing number of businesses, and with the database systems getting more powerful and richer in functionality, the stress on corporate database administrators has gown exponentially in the recent past. We hope that this paper will eliminate some of that stress and at the same time enhance the efficiency of their database systems.

ACKNOWLEDGEMENTS

The ideas in this paper were developed with the help of many experts at Oracle. Their contribution is greatly appreciated. Special thanks are due to Graham Wood, Benoit Dageville, Alex Tsukerman, Ron Weiss, and Sushil Kumar for their input and feedback.

REFERENCES

1. J. Loaiza. *Optimal Storage Configuration Made Easy*, http://otn.oracle.com/deploy/performance/index.htm
2. S. Kumar. *Server Manageability: DBA's Mantra for a Good Night's Sleep.*
3. G. Wood, C. Dialeris. *Diagnosing Performance Using StatsPack*, http://otn.oracle.com/deploy/performance/index.htm
4. Oracle White Paper: *Oracle Performance Improvement Method*, http://otn.oracle.com/deploy/performance/index.htm
5. T. Lahiri, *et al*. *Fast-Start: Quick Fault Recovery in Oracle*, http://web06-02.us.oracle.com/deploy/availability/techlisting.html
6. T. Bednar. *Database Backup and Recovery: Strategies and Best Practices*.
7. Oracle9*i* Database Administrators Guide, Release 1 (9.0.1)
8. Oracle9*i* Database Performance Guide and Reference, Release 1 (9.0.1)
9. Oracle9*i* Database Concepts, Release 1 (9.0.1)
10. Oracle9*i* Performance Methods, Release 1 (9.0.1)
11. Oracle Technology Network, http://otn.oracle.com