Application/Database Tuning – A Practical Business Approach

Sheilah Scheurich
The Consulting Team/ARIS Corporation

Introduction

This paper will enable the DBA and programming community to effectively tune the Oracle Applications to meet the business requirements of the user community. With case studies, scripts and Oracle provided tools such as explain plan, tkprof, and other utilities, this paper will lessen the reactive nature of applications support by developing proactive methodologies. HRMS and Financials including concurrent manager troubleshooting are covered.

Often, tuning associated with applications has focused on the Information Technology arena and not on the business side of a company. Unfortunately, if there is not a full understanding of how the system is used, then complete and effective tuning cannot occur. This paper will focus on how to work with the application and in particular, business needs. Areas of potential tuning are operating system, database layout and database tuning. Due to the scope of this paper, the operating system will not be covered.

"It's the application that is wrong."

When using standard performance tuning methodologies, the DBA and programmer should start with the application and work their way up. However, with the advent of off the shelf software, this methodology must change. Like or not, changing the code within the application is inappropriate and it could jeopardize the support of the application. Changes to the application code could also detrimentally affect the accuracy of your payroll or accounting functions. Imagine having to tell management that the numbers they just reported to the SEC may not be accurate due to software modifications. Simply put, do not do it.

The best way to evaluate customizations is during the development phase. While this is considered a best practice recommendation, many programmers pay only lip service to the process. Two other factors can affect the development process, time, and experience. Reading this paper will not provide you with more time, but it can save the developer and DBA time by not having to troubleshoot performance problems later once the software has been implemented.

Oracle Applications uses the rule-based optimizer. Therefore, there are standard ways to write customized software. The rule based optimizer looks at the access path to determine which will be the most optimal path to take in order to be efficient. They are ranked and the optimizer will take the first access path it finds. The lower down on the list of rules, the slower the query will most likely be. The access paths are ranked in the following order[i]

1. Single row by ROWID
2. Single row by cluster join
3. Single row by hash cluster key with unique or primary key
4. Single row by unique or primary key
5. Cluster join
6. Hash cluster join
7. Indexed cluster key
8. Composite key
9. Single-column indexes
10. Bounded range search on indexed columns
11. Unbounded range search on indexed columns
12. Sort-merge join
13. Max or Min of indexed column
14. Order by on indexed column
15. Full table scans.

A few things worth noting here. Notice that when searching for minimum and maximum values, they run more optimally if indexed. The same is true if using order by clauses. This does not mean to add indexes to improve performance. What it means is to try to write customizations that utilize existing indexes.

The following query will allow you to find the columns. The position is important because this should be the way that the columns are referenced in the where clause.

Select column_name from dba_ind_columns where index_name = '[INDEX_NAME]' order by position;

If you do not know what indexes are used on the table use the following query:

Select index_name from dba_indexes where table_name = '[TABLE_NAME]';

One other important issue when writing software – be sure to include ALL the columns of an index in the where clause. Also, ensure that the indexed columns are the leading part of the where clause. The easiest way to determine which index is being used is to use the Oracle provided tool **tkprof.** Many programmers prefer the use of EXPLAIN PLAN. While effective in determining which indexes are used. They are not effective in determining programming errors such as Cartesian joins (also known as Cartesian product). On rare occasions, the need for indexes is indicated. More often that not the programming errors consist of Cartesian joins or the improper use of indexes.

What is a cartesian join? A cartesian join all the rows of the tables with each of the rows of both tables. For example: Table A is defined as: TABLE_A (a_value varchar(1)). It has the values of A, B, and C. Table B is defined as TABLE_B(a_value varchar(1),b_value number(1)). It has the values of A,B,C and 1,2,3. Using the select statement:

SELECT B.A_VALUE,B_VALUE FROM TABLE_A A,TABLE_B B WHERE A.A_VALUE = 'A' AND
B.A_VALUE = 'A' ;

The result will be

A_VALUE	B_VALUE
A	1

The results are correct, however, rather than looking at only one row of data, the database looked at 9. It will look at the sum total of rows of TABLE_A * TABLE_B. While this is a simple example, the number of rows evaluated could be substantial in a large table in Financials or Payroll.

Normally, the programmer is going to be able to easily determine that there is a problem, simply because of the large number of rows retrieved. Another way, once familiar with the fact that these exist, is the length of time that the queries take. This takes experience, until this, the programmer should rely on the tools such as tkprof and explain plan. As explained earlier, many Cartesian products will return a large number of rows. However, occasionally, the results from a Cartesian product will produce the correct results. Due to the limited amount of data available to the programmer, unless the programmer specifically checks for such a problem, the software will be released. Below is an example of a Cartesian join in the form of a **tkprof** trace.

Two clear indicators will show the programmer that a Cartesian join exists. The first is the high number of times the data is accessed in memory. This indicates that the system is looping through the data repeatedly. Secondly, and the most obvious, the high number of rows that are being searched. The table only has only a few rows and the number of rows accesses is significantly larger. If the number of rows can be divided by another and the result is the actual number of rows in the table, then this is proof a Cartesian product join.

Interpreting TKPROF results

There is a drawback to using tkprof. It can be confusing. Another problem is that many people believe that you have to have statistics on in order to get real meaning from the output. This could not be more wrong. The primary purpose for utilizing tkprof is to determine the effectiveness of a specific statement or process. The non-timed statistics provide enough information to determine where the problems lie. Below is a tkprof output of a suspected Cartesian product.

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0	0	0	0	0	0
Execute	1	0	0	0	0	0	0
Fetch	1	0	0	0	0	0	0
total	1	0	0	100	12329	6	628

Misses in library cache during parse: 0
15 user SQL statements in session.
0 internal SQL statements in session.
15 SQL statements in session.

The columns are defined as the following: count is the number of times a query is parsed, execute and fetch. CPU and elapsed times are not used unless statistics are on the database. Disk represents the number of data blocks, and query represents the number of buffers in memory that are being accessed. Notice the number in the query column. This means that the data within memory was accessed repeatedly. The current column is the rows retrieved for update and insert. The rows are the number of rows retrieved in the query. Notice the high number of reads compared to the relatively low number of rows retrieved. Clearly, this SQL statement can be improved.

Other Issues

Pinning packages to improve Performance

Oracle recommends that packages be pinned into shared memory to help performance. While this may be helpful for sites without a great deal of memory, research has proven that in systems with large shared pool sizes, the packages are loaded upon the initial access and never leave memory until the database is shutdown. This can be proven by examining the table V$DB_OBJECT_CACHE. Note the number of loads verses the number of executions of this table. If the number of executions is high and the loads low, then this may prove that pinning is an exercise that may prove unnecessary. The packages are most likely not aging out.

There are some exceptions to this. If a system is analyzed nightly, then it may be necessary to pin the packages back into memory in order to ensure that the initial users of these packages will not see a performance problem.

Pinning is the process of placing an object into shared memory. This prevents the object from aging out. According to the documentation within the actual procedure (DBMS_SHARED_POOL), the process of pinning a package may become obsolete.

There are several ways to pin packages. The way that I like to determine which package to pin is to look at the table DBA_DDL__LOCKS. Note the packages that are pinned. Then pin the ones that are most used. Another way to find which packages to pin is just by looking for the large ones. Use the package dbms_shared_pool.keep. The syntax for pinning the package DBMS_STANDARD would be:

SET SERVEROUTPUT ON SIZE biggest
dbms_shared_pool.sizes(biggest IN NUMBER)

Once the packages have been determined, then use the following command to pin the packages. The command for pinning the package STANDARD will be:

execute dbms_shared_pool.keep('STANDARD');

One thing to be careful of, you can actually fill your shared_pool with nothing but pinned packages. This will be identified by the error **ORA-4031.**

Bind Variables

One of the reasons that the hit ratio can be artificially inflated is due to the unique qualities of an application. With bind variables, particularly the ones from forms, many queries will remain in memory as the same SQL. This can artificially inflate the hit ratio of the database.

Why is the use of bind variables important? The reason is in the way that Oracle interprets SQL statements. When a statement is sent to Oracle, it must be parsed and placed into the library cache. However if bind variables are used, then this statement will not be reparsed or takes up more space in the library cache. For example in the statement:

Select emplid from employees where emplid = 123;

Each time the user wants to look for another emplid, the Oracle system will recognize this as a new statement. If however, this query is placed into a form, then the form will use bind variables and the query will look like this:

Select emplid from employees where emplid =:1;

When writing SQL statements to utilize the efficiency, there are a few other rules to note. Note the case of the statement and the way a statement is written. The following statement:

Select 'X' from dual;

Is different than the statement:

Select 'X' from DUAL;

There are things to note here. When building dynamic statements or writing code that will use the same statement repeatedly, put the statement in a procedure or trigger, or be absolutely sure that the statement is written the same.

This will not be as big an improvement as increasing the block size or db block buffers but it can help because the statement will not be reparsed.

Another IO intensive operation and potential problem is the operating system swap files. This will be developed with the cooperation of the system administrator. The amount of disk swapping will depend upon memory availability and the type of operating system used. Check with the system administrator to

determine if the IO activity of the swap space should be considered when laying the database out.

Other issues that the DBA must also be aware of areitems not associated with the database. These can directly affect the performance of the system. For example, an area of contention that is often overlooked, particularly in a financial system, is the location of the application outfiles and the log files. These particular files are extremely IO intensive and can dramatically slow the system down if placed on an equally IO intensive part of the database. These files are commonly located in the $APPL_TOP/common/ directories. If the performance of the online system is poor overall, then the problem most likely lies within the system rather than the application.

Database Optimization

Several areas within the database can significantly affect database performance. The best place to start is at the beginning. Two things can help the performance before the first query is ever performed on the system. First is database layout.

Database Layout

While Oracle recommends that there be around 20 or so disks used in the optimal layout of a system, most organizations do not have the luxury to utilize this large number of disks, particularly if the system is to be mirrored. Some DBAs like to have drives that are hand striped rather than utilizing the hardware or operating system perform the same function. This should be avoided if possible. Efforts by the DBA will not be as successful as an operating system (good) or hardware striping (preferred). The reasoning is simple; most of the companies, both hardware and software vendors have invested a great deal of time and money in the algorithms used in striping the system. Striped disks are covered later in this paper.

Another portion of the database layout that should be performed in the beginning is the layout. At a minimum, the index files and the data files should be placed on different raid sets. Ideally, each application should be placed on individual raid sets. For example, Accounts Receivables datafiles should be put on one striped set and the indexes on another. However, this can be cost prohibitive. In addition to the datafiles and index files, another set of files to consider when laying out the database is the redo logs and archive logs. If possible, these should be placed on static drives.

In some of the performance tuning books, there is a recommendation that the DBA should stripe the redo logs. One of the primary reasons for creating striped redo logs is because of the write processes. Stripe sets of high granularity (small block sizes) should write faster because the process of writing will now cover multiple disks. There is also another reason for striping. When a computer has multiple processors, it can have multiple redo copy processes. Because more than one process is actually writing to the same redo log at the same time, striping will enable more processes to access the file at the same time. However, it should be unnecessary in case of a single CPU environment. There is one other performance consideration when determining if you should stripe redo logs. Is the performance gain by striping logs be significant enough to justify the removal of spindles from other database files (data and index files). In essence, redo log striping is good only on a system where unlimited disk (spindles) are available. The amount efficiency gained by removing spindles from the datafiles or index file in order to stripe the redo logs (on a single CPU system) is most likely not cost effective.

When trying to determine the cost effectiveness of striping redo logs, the DBA must take into account that each of the redo logs is mirrored. Each of the disks that contain the redo logs will have the potential of being as IO intensive as the next. Therefore, they will be creating contention on each of the striped volumes the redo logs reside on. Great care must be taken to ensure that the redo logs will not cause IO contention problems with the rest of the database.

Most DBAs know about the problem concerning the **redo log space requests**. This is a parameter that can be found in the V$SYSSTAT table. It should be zero. If it is not, then there is a performance problem. The issue is not in the value, rather how big that value is. If it is up in the thousands and the database has only been up for only a week or so, there is a larger problem than simply increasing the log buffers parameter in the init.ora. The redo logs are most likely spinning. This occurs when the redo logs are writing and switching at such a rate that before the first one has completed writing, the logs have switched back to the original one. This is de-

termined by looking at the alert.log. Below is an example of spinning redo logs:

```
Fri Jul 18 16:02:04 1997
Thread 1 cannot allocate new log,
sequence 8844
Checkpoint not complete
Current log# 3 seq# 8843 mem# 0:
/u04/oradata/PROD/redo03.dbf
Current log# 3 seq# 8843 mem# 1:
/u05/oradata/PROD/redo03b.dbf
Thread 1 advanced to log sequence
8844
Current log# 2 seq# 8844 mem# 0:
/u02/oradata/PROD/redo02.dbf
Current log# 2 seq# 8844 mem# 1:
/u03/oradata/PROD/redo02b.dbf
Fri Jul 18 16:02:45 1997
Thread 1 advanced to log sequence
8845
Current log# 1 seq# 8845 mem# 0:
/u02/oradata/PROD/redo01.dbf
Current log# 1 seq# 8845 mem# 1:
/u03/oradata/PROD/redo01b.dbf
Fri Jul 18 16:03:00 1997
Thread 1 advanced to log sequence
8846
Current log# 4 seq# 8846 mem# 0:
/u06/oradata/PROD/redo04.dbf
Current log# 4 seq# 8846 mem# 1:
/u07/oradata/PROD/redo04b.dbf
Fri Jul 18 16:03:15 1997
Thread 1 cannot allocate new log,
sequence 8847
Checkpoint not complete
```

Figure 1 - Example of alert.log with spinning logs

Notice the following: The checkpoint not complete error indicates that the system had to wait until the original redo log had completed writing. Another (and equally important) issue to notice is the amount of time that the redo logs are switching. These redo logs are switching at a rate of every 15 seconds! Adding additional redo logs will only allow more redo logs to write, but the read – write heads will still be down all the time. One way to fix this is to increase the redo log size substantially. I start with a size of 50Mg and will increase it based on the number of users and DML/DDL activity on the system. On a large financial system with a couple of hundred users during closing a size of 100M or more is not unreasonable. If you as a DBA are concerned about recoverability, simply set a checkpoint to occur at about half of the redo log size and during a specified time. Then, even when the system is relatively quiet; you can be assured that the recovery time will not be excessive.

Checkpoint Tuning

This is another area of tuning can significantly improve performance. One great way to determine if the checkpoint process is working as well as possible is to set the init.ora parameter **log_checkpoints_to_ alert** to true. Then monitor the alert.log to determine if the checkpoints are occurring unreasonably. If you are comfortable with the checkpoint occurring only during a redo log switch, then set the parameter **log_checkpoint_ interval**, to a size larger than the redo logs. Another word of caution here, this parameter is not based on database blocks or bytes. It is based on operating system blocks. In most Unix systems, this is normally 512k. Be aware that this will mean that the checkpoint may occur as infrequently as once every couple of hours if the database is quiet. Do not tune to accommodate this quiet period! By doing so, the performance doing a close or payroll process will suffer. If the recovery issue bothers you, set the parameter **log_checkpoint_timeout**. This value is set in seconds.

Database Block Size

Oracle recommends that the block size of the database should be 8k. More systems are becoming extremely large and this standard may well be worth investigating. Initial investigations indicate that even a larger database block size of 16k may help the performance of such large systems. However with this aside, many times even the 8k-block size is not used. This is primarily due to building the database with the default. On most UNIX based systems this will default to 4k. This should be avoided at all costs. This can be avoided by simply placing the block size in the first init.ora file PRIOR to building the database. Once built, this value cannot change. A word of warning, unless the block size is specifically set during the initial startup, then it is too late. This can be particularly difficult in systems such as VMS where you have to use the orainst program to build the database. If possible, use scripts rather than this tool to control the block size.

Session Vs System Hit Ratio

Every DBA to check the database block buffer hit ratio. Some DBAs will say that a hit ratio of less than 90% are poor. I believe that anything less than 95% is poor. Anything that will take the entire database hit ratio down by 5% should cause concern. While this may be considered wasteful, with the abundance of memory in most systems, high db_block_buffer

sizes are recommended. The system block buffer cache hit ratio while important, should not be the sole SGA statistic. Nor necessarily the most important.

The DBA is trained to look at the entire system. With applications, the overall health of the system may not be as important as the health of the specific sessions used to support the business. For example, in a financial system the system will remain relatively quiet until closing. Then the system will be completely crunched with the constant submission of FSG's, interfaces and consolidations.

Tuning should focus on specific processes that support the business. It is easy to tune for the transaction based processes such as data entry. The batch processes should be of greatest concern. If tuned properly, the online transactions will be tuned simply when tuning for the whole system.

The following script will identify those sessions that are lower than 90%. The overall hit ratio should be at approximately 96 to 99 %. It is most likely that most of the sessions (more than 90%) are at 99% or 100%.

```
select s.sid sid,v.serial# serial#,v.status,
to_char(sum(decode(name,
'physical reads',value,0)),
'999,999,999,999') "physical reads",
to_char(sum(decode
(name,'db block gets',value,
'consistent gets',value,0)),'999,999,999,999')
"logical reads",
to_char(((1-(sum(decode
(name,'physical reads',value,0))/
sum(decode(name,'db block gets',value,
'consistent gets',value,.0000001))))
*100),'9999999')||
'%' "hit ratio"
from v$sesstat s,v$statname n ,v$session v where
s.statistic# = n.statistic#
and s.sid = v.sid and
v.type != 'BACKGROUND'
group by s.sid,v.serial#,v.status having
((1-(sum(decode(name,'physical
reads',value,0))/sum(decode(name,'db block
gets',value,'consistent gets',value,0))))*100)< 90
```

Few sessions with percentages less than 30% are most likely the cause of poorly written customizations. As much as most people like to complain about Oracle Application Online Performance, I have rarely found their online transaction software causing performance problems with a low hit ratio unless the database was sized improperly.

Init.ora parameters

Below are some recommendations for init.ora parameters that can affect the performance of the Oracle Applications.

Cursor_space_for_time

Set this value to true in order to improve performance, particularly with the online system. This parameter prevents any SQL statement in the library cache from being deallocated as long as the application cursor is open. Since most of the queries within the application use bind variables, this will mean that the more frequently used forms will never be closed or removed from the library cache. This parameter is particularly helpful in the HRMS systems. Because much of the access is online, and related to the same form or sets of forms, keeping the cursors open can help. This improves performance, however done

Be careful, by setting this value to true, the number of open cursors will have to be increased. This will also increase the amount of memory. If this parameter is set there is the potential that the number of open cursors will be exceeded. If this happens, the following error may happen:

ORA-01000 maximum open cursors exceeded

This error will most likely occur when multiple users are accessing the system. To proactively manage the system, it is better to simply increase the maximum open cursors parameter when you set **CURSOR_SPACE_FOR_TIME**.

CHECKPOINT_PROCESS
Set this value to true if you have more than 15 or so files. This relieves the logwriter process from performing checkpoints. In version 8, this parameter goes away and is set automatically.

DB_FILE_MULTI_BLOCK_READ_COUNT
When Oracle is performing a sequential read, this is the largest amount of data the database can retrieve in a single read. This parameter should be set to a level of 32 or higher. This is particularly effective during initial reads. The more memory available in the dictionary cache, the larger this parameter should be. This parameter cannot be set larger than the operating system maximum IO size / db_block_size.

DB_FILE_SIMULTANEOUS_WRITE
This parameter is the write mate of the above parameter associated with the read. The maximum

value is 24. However, if the operating system does not support more than one write per disk, then the value should be set to 1. Ideally, this value should be set to no more than 2 times the total number of disks available.

PRE_PAGE_SGA
I automatically set this parameter to yes. While it affects the performance during startup, it removes the paging process as the database tries to acquire more paging space as needed. This is only true with systems with abundant memory.

Library Cache

This is one of the most important areas that can affect the performance of the system – particularly online performance. One of the more powerful tables to use in investigating the effectiveness of the current settings are V$LIBRARYCACHE, and V$ROWCACHE By utilizing the following queries, the ratio of pins to reloads should be determined. Some of the areas of statistics and values to look at are PINS and RELOADS. There is only one issue concerning a ratio that cannot be determined. In the V$LIBRARYCACHE, there is a column called invalidations. Invalidations occur when one of the objects in the librarycache have been changed. This invalidation process will result in a reload. If there are a high number of invalidations then this number should be accounted for in the ratio. Oracle recommends that the RELOADS value be as close to zero as possible. However, I do not believe that this is set in stone like the value REDO LOG SPACE REQUESTS. When evaluating the reloads, you need to determine if the reloads are occurring due to an inadequately sized shared pool or that there are very large FSG's running. These can be extremely large and will age out if used infrequently. Because of this, the DBA should be familiar with the application as well as the database. If a user is running an FSG infrequently, the fact that it is aged out may not be a bad thing. By aging out, other, smaller queries can be kept in the library cache. In this case, a large database block buffer should be set high in order to keep the tables accessed in memory.

Shared Pool Fragmentation

There is another problem with shared pool – fragmentation. This occurs when large packages are aged in and out and finally prevent even smaller packages from being kept in memory. How can you tell if this is happening? The easiest way is if the system is faster the first few days after a reboot of a database and slowly gets worse. Another way is if the error **ORA-4031** occurs. If this becomes a problem, the best way to fix this without rebooting the system is to clear the shared_pool and then pin the larger packages. The command for clearing the shared pool is:

Alter system flush shared_pool;

Rollback Segments

Many database tuners will recommend that the DBA set up a rollback segment dedicated to large transactions. I disagree with this methodology. I believe that the rollback segments can be uniformly sized to meet the application and online transactions and large batch processes. By sizing all the rollback segments uniformly that will, when expanded out, meet the needs of the large processes. The following is a way to accomplish this: To determine the size of the rollback tablespace take the

1. Create a large tablespace for rollback segments. To determine the size of the tablespace, take the sum of the data and indexes of the largest tablespace. While excessive, this will ensure that the database will support even the largest DML process.
2. Multiply that by .75. This will ensure that in the event that multiple large batch processes are occurring, there should be sufficient space to support them.
3. Divide the value determined in step 2 by an even number, I prefer 100. This is the initial and next extent for your rollback segments.
4. Set the minimum extents to 20 to support most batch processes.
5. When establishing the optimal be sure to account for the initial extent in addition to the minimum extents.

Now the database should support any transaction that should occur in the database.

Spindles, Spindles, Spindles

When sizing for the database and the application, forget for a moment the amount of disk space. In this case, size does not matter. While the statement can be humorous, it is very serious. Often one of the biggest mistakes in developing an application layout is to concentrate on the amount of disk space and not the amount of disks. The number of disks (also known as spindles) will determine how well the database (and ultimately the application) will run. In many com-

puter systems, the smallest disk available is now 9 gig. In maximize performance on a database system, the disks should be striped (also known as RAID). The fastest striped environment is known as RAID level 0. What this means is that disks are logically (or depending upon hardware – physically) tied together as a single disk. Ideally, this should be between 4 and 5 disks. This means that at a minimum, the layout of the disks will be 36 gig. However, this does not include the proper layout. Using the standard performance tuning methodology of separating the data from the index files, this will increase the available disk space to 72 gig. While this can be a phenomenal amount of disk space especially for a smaller database environment, however, when compared to the relative amount of money spent on software, implementation and consulting, hardware is a very small part of the overall budget. Unfortunately, when trying to save money, this important part of the performance puzzle is sacrificed. RAID 0 while extremely fast, has one very large drawback. Your data and system is unprotected. You are safer running naked through a cactus field than you are running on RAID 0. The next fastest method is RAID 0 + 1. This is RAID 0 but mirrored. This is most often the fastest next to RAID 0. However, this really depends upon the number of controllers that exist on the system. If each RAID 0 set is mirrored on separate controllers, then the system will be considerably faster. However, this is not a complete system. Other tablespaces, such as SYSTEM, ROLLBACK, and TEMP should also be isolated. TEMP while important, may not have to be isolated as originally thought. This is primarily due to the large memory now available on most systems. There is a limit on the size of the SGA on 32 bit systems. However, this limit is currently 3.8 gig. At this size, very little of the TEMP tablespace will be used. While this does not mean that the TEMP tablespace will be completely quiet, if the SGA is sized properly, then it should not be as considered a major tuning problem if disk space or spindles are limited. In most environments, the TEMP tablespace can be placed with SYSTEM tablespace, and while not perfect, it can be placed with the ROLLBACK. However, TEMP should not be placed with INDEX or DATA tablespaces. This is to prevent the possible disk contention associated with large sorts and accessing the data associated with the sorts.

Application Specific Issues

Payroll

The layout of a database will normally not change significantly on the application. However, the way that the system is managed can change significantly. Payroll systems will run much differently than those of the financial systems. For example. In a payroll system, if the payroll runs only one payroll, then much of the tuning efforts should be focused on this single batch process without adversely affect the online processing. As stated before, tuning for the batch processes should normally help the online transactions, it may not be the complete answer.

The application software cannot be modified. The addition of indexes may help, but the addition of indexes is not recommended as a normal tuning practice. While the index may help a single process, it can adversely affect other processes. This should be done on a test environment before moving into production.

While many people will contend that the application software is not optimally tuned, this is not the case for most of the online transactions. As stated before, the bind variables allow the Oracle system to see each statement as the same. If a person performs accesses the job record associated with one person, the next time another person is accessed, Oracle will think that this is the same query. Therefore, the query will remain in the shared pool area and will not require reparsing. This can help improve the performance of the system.

The effort should be placed on evaluating the batch processes associated with the payroll. This does not mean adding additional indexes or other modifications to the code or structure to the database. An emphasis should be placed on the use of the memory. Look at the hit ratio while the payroll process is running. This is when the system should be tuned. The DBA can use the utbstat and utlestat to determine where the changes need to be made. This will be easily determined if the payroll process is effectively using the memory. Look at alert.log during this same process. Note the frequency of the log files switching. The log files while normally not a problem can effectively shut down a batch process such as payroll if not sized for the specific application. In the case of multiple payrolls other areas that will dramatically affect the performance is rollback segments.

Financial

As stated previously, the perfect environment would be to separate out each product and each index tablespace on separate raid drives, however, this may not be possible. Because of this, the best way to manage these processes is to layout as many different applications as possible. The Accounts Receivable tablespace should be kept away from the General Ledger tablespaces. Project Accounting, a uniquely different (and intensive process) is not included in this suggested layout.

Financial systems are primarily batch in nature and intensively trigger and procedural heavy. This phenomenon places the DBA in the unique position of trying to allocate as much memory for library cache as there is in the dictionary cache. These batch processes are massively IO intensive and are used for a relatively short period of time (in terms of intensity). This means that a database system is tuned for only a week or so of activity. Tuning the system so that it runs acceptably during the other times and slowly during the close is NOT MEETING business needs. This should never be the case in tuning. Always tune for business needs and not overall performance.

Another thorn in the side of performance of Financial Systems is the misuse of FSG's. While these reports represent a crucial part of the business functions, they should be isolated and ran (if possible) at non-peak times. Another issue is to ensure that these are placed on their own concurrent product manager. The concurrent management issues are discussed below.

Concurrent Manager Troubleshooting

While there are specific, things can be done to actually improve the performance of the system; the biggest problem in performance tuning of the concurrent managers is perception. There is a huge difference between hung and slow. When concurrent managers "pend" out, this is a result of something blocking the other processes. When the system is slow, there is some movement. The user community associates seeing the "pending" status as a system being slow. It may simply be nothing more than the need to add additional threads to that particular manager. Alternatively, the solution may be to add additional managers.

The most common issue associated with concurrent manager performance is the lack of threads available for the processes. Creating a concurrent manager for FSG's and limiting the possible paths of these to only 3 to support a user community of 200 or more is unrealistic. While there is not any pat answer, the number of concurrent manager threads should be capable of supporting the user community. Often the numbers of threads are limited due to the concern of creating a poorly performing Financial System. The truth is, that rarely does performance degrades to the point that the level that those responsible for the threads believe that it will. Increase the threads until the system starts to be affected. This is the user community's system – let them tell the administrator when performance is unacceptable, not the other way around.

Another problem associated with the concurrent manager is locating the errors or "silent" abends. Often, the system administrator or the DBA will be tasked with the responsibility of troubleshooting concurrent manager problems. This is due primarily to the inaccessibility to see the log files on all of the managers and processes. All log files originate on the server. Access to the server may be limited to a select few. Because of this, they will also be asked to assist in troubleshooting problems that may appear in these logs. Some of the reasons for access the operating system files are:

- The System administrator or DBA may not have access to the client application software.
- The user is unable to see the actual error message due to improperly install log directories
- Application installation and Patch application.

One of the pitfalls associated with the concurrent manager process is not properly placing the directories or defining them in the environment files. Should this occur, the user may be looking at concurrent manager files that reference older or different concurrent processes. This can be seen when the user complains that the process submitted failed but that the status or the log file indicates a success.

When troubleshooting application problems, particularly ones where programs, output files or logfiles are going to the wrong place, be sure to look at all the env files that exist on the system. Most DBAs or System Administrators are aware of the env files located at the top of the application directory (also known as APPLTOP).

Other Environment Files

Two additional env files can jump up and bite. These do not affect the performance or the ability to trou-

bleshoot the database, application, or concurrent manager process, but can significantly affect an installation, addition or upgrade of the application one is a little file called fndenv. This is located in the FND_TOP directory. It contains the environment setup for the Application Library Setup. It can also have in its definition the operating system mask for the application. These properties can affect the protections of a file. This particular "feature" can jump and byte many a frustrated installer or implementers. This also defines the extensions of each of the other applications such as plsql, SQL, and others. Here is an example of this file:

```
# $Header: fndenv 61.0 96/10/28 19:45:05 porting ship
#================================================
#  Copyright (c) 1988 Oracle Coporation Belmont, California, USA
#                                All rights reserved.
#================================================
# FILENAME
#            fndenv
# DESCRIPTION
#            Appliction Object Library environment setup
# NOTES
#            Requires APPLTOP to be set
#================================================

#      base  directory  and  "extensions"  (apply to all products)

            APPLBIN=bin;                  export APPLBIN
            APPLCFRM=cmforms;             export APPLCFRM
            APPLHLP=helptext;             export APPLHLP
            APPLINC=include;              export APPLINC
            APPLLIB=lib;                  export APPLLIB
            APPLMSG=mesg;                 export APPLMSG
# Define the following value to enable a file protection mask
# other than that defined in $FND_TOP/$APPLBIN/startmgr
#           APPLMSK=022;                  export APPLMSK
            APPLORB=ar25runb;             export APPLORB
            APPLORC=ar25run;              export APPLORC
            APPLPLS=plsql;                export APPLPLS
            APPLREG=regress;              export APPLREG
            APPLRGT=regress;              export APPLRGT
            APPLREP=srw;                  export APPLREP
            APPLRPT=rpt;                  export APPLRPT
            APPLRSC=resource;             export APPLRSC
            APPLSAV=save;                 export APPLSAV
            APPLSQL=sql;                  export APPLSQL
            APPLUSR=usrxit;               export APPLUSR

# Call devenv to set up development environment

        . ${FND_TOP}/${APPLUSR}/devenv
```

Figure 2 - Example of fndenv file

The next file, devenv, is located in the $FND_TOP/$APPLUSR determines the location of library files, in particular, the Oracle Receivables and the Oracle Payroll. While this does not directly affect the overall performance of the database or application, it is important to know that these files exist and may hamper future upgrades or the addition of new modules. The file is too long to include in this paper, however the biggest issue is to make the DBA aware that this file exists and to ensure that the paths defined in these files are relevant to the

Case Study

Programmer Sam made a modification on a procedure to meet the requirements of the addition of a new table within the interface process of the Oracle Payroll. When testing the software on the development database, the program provided accurate information. Therefore, the modification was moved into production. When installed, the performance of the program degraded to a point that the program was killed each time because it simply would not finish. Each time, the program was run for a minimum of 12 hours before being killed. Upon evaluation, it was determined that the addition of the new table created a Cartesian join. When the programmer moved each of the joins into a separate cursor, the program was reduced to 3 minutes.

Case Study 2

A DBA had a process that would analyze a schema on a bi-weekly basis. After this biweekly process, the system performance particularly with large queries was significantly slower. However, when the analyze was performed prior to the restart of the database, performance improved. Perplexed, several areas were investigated to determine what could be causing the problem. After a few tries, the DBA discovered that if he used the command:

Alter system flush shared_pool;

The system significantly improved. Therefore, this was included in the future analyze process.

Conclusions

Leave the main application alone. It is better to focus on any customizations and leave the application tuning to Oracle. Focus on tkprof and other tuning methodologies in order to ensure that the customizations are written correctly.

IO in all areas of the application must be evaluated, not just those items associated with the database, but the operating system (such as swap files) and application out files.

Database layout is one of the biggest performance enhancers of the system. Focus on the number of disks and not on the disk space. The more spindles (disks) the better performing the system.

Focus on session activity rather than overall system activity. Focus on business practices and tune for the days the system is used and not for the overall use of the database.

Concurrent manager problems are more often due to application setup and not just related to performance.

Perception of the system, such as pending processes are just as real as if the system were slow. Education of the user community is necessary in order to help this problem.

About the Author

Sheilah Scheurich has been working on Oracle based systems since 1989 and version 6. Her experience includes development, programming, and database design. She has been working with Oracle Applications for two years. This includes Oracle Financials, Oracle HRMS, payroll, and Oracle Time Management. She currently works for The Consulting Team, a division of the ARIS Corporation as a consulting DBA with an emphasis on tuning.

[i] Oracle7 Server Tuning – The Optimizer: Overview