

Why am I waiting? Oracle Response Times on HP Servers

Adam Grummitt and Tim Foxon

Metron Technology Limited, Taunton, U.K. (action@metron.co.uk)

The response times provided by applications based on ORACLE databases are determined by a large number of factors. It is quite possible (and, indeed, common) to achieve highly variable and/or very poor end-user response times, even when the very latest high-performance hardware and system software is installed.

This paper describes the ORACLE architecture, and the way that it interacts with HP/UX from a performance management viewpoint. The factors which contribute to response times - including the use of logical as well as physical resources are described. The really important metrics to look for, and those actions that can have a measurable (and therefore worthwhile) effect, are defined by reference to real case-study data.

The paper concludes with a description of the way in which data from ORACLE and HP/UX may be used to predict the way in which response times will be affected by changes in the workloads and/or configurations. A simple and practical method using analytical modelling techniques to achieve this will be described.

1. Introduction

Embarking upon a performance management exercise in an Oracle environment under HP/UX can be a daunting prospect. Both Oracle and HP/UX are blessed (?) with a plethora of performance metrics and configuration parameters. However, approaching this topic by listing all the available performance metrics is like teaching a language by supplying a dictionary. In contrast to the static view provided by a list of metrics, section 2 of this paper provides an assessment of the dynamics of an Oracle system. From this abstraction the areas that should receive attention are derived:

- major SGA structures and their effect on physical resource consumption (section 3)
- disk storage and fragmentation effects (section 4)
- SQL and database navigation (section 5)
- contention for logical resources (section 6)

The paper concludes with a discussion of the desirability and methods of modelling Oracle systems.

Each section is illustrated with reference to a particular performance management case study. The application was implemented on an 4-processor HP/UX machine with around 250GB of striped filestore on over 60 physical devices. The users of the application experienced highly variable response times that, at peak periods, were limiting their throughput. We were asked to assess and remedy the causes of this, and to define the hardware resources required to accommodate the expected increase in end-user work.

When assessing the performance of an application it is essential to address the system as a whole, rather than just from an Oracle or an operating system viewpoint. A recent performance study illustrated the dangers of a restricted view. We were asked to identify the reasons why a particular Oracle application was performing badly. The Oracle team had attempted (quite correctly) to

maximise the database buffer cache hit-rate. By increasing the number of buffer blocks they were eventually able to achieve a hit-rate consistently in excess of 99%. Unfortunately this increased the size of the SGA to the point that UNIX started to page. This caused considerable amounts of I/O (thus adversely affecting performance) which was not apparent from any of the Oracle metrics.

When tuning any Oracle system, the areas to address (in this order) are:

- (a) the design (if it's not too late!)
- (b) the application
- (c) memory
- (d) I/O
- (e) contention

The main goals in Oracle tuning (taken from Oracle's own documentation) are to make sure that:

- SQL statements access the smallest possible number of Oracle blocks
- if a block is needed, it is cached in memory
- users share the same code
- when code is needed it is cached in memory
- where physical I/O to disk is unavoidable, it is performed as quickly as possible
- users never have to wait for resources used by others

The ways in which these goals can be achieved are discussed in sections 3 to 6 of this paper.

2. Oracle Dynamics

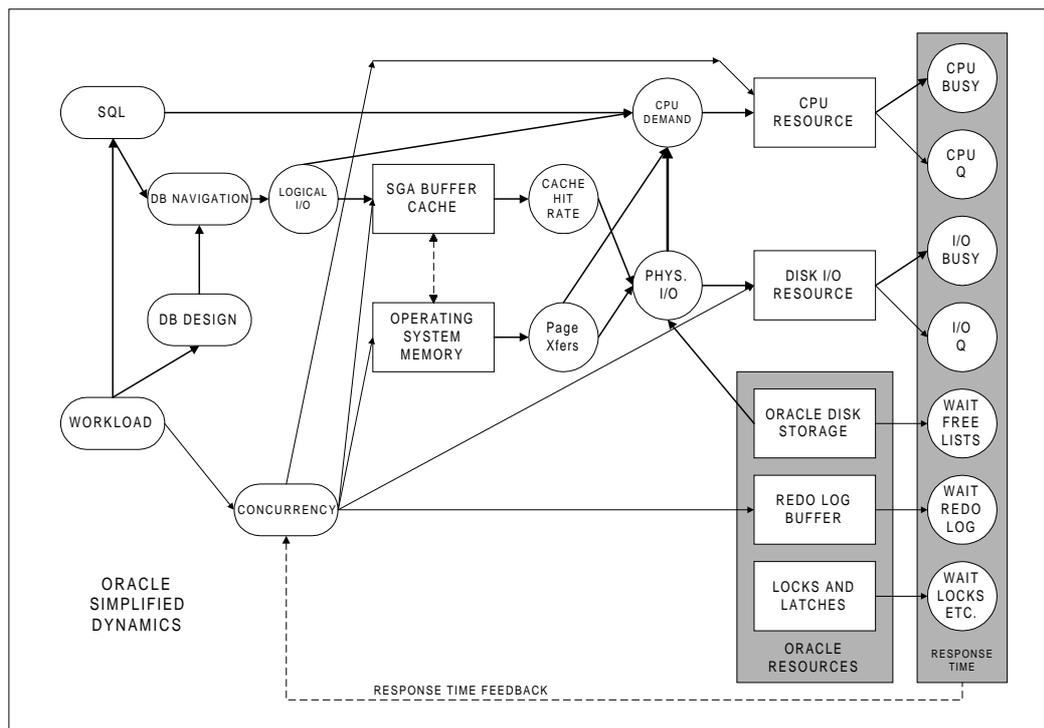


Figure 1. Oracle Simplified Dynamics

Figure 1 shows, in a simplified form, the main Oracle dynamics. This picture shows the relationship between an occurrence of an Oracle workload, and the factors that can affect its response time. The oval shapes represent application requirements, the square shapes indicate physical resources and the round shapes performance metrics. Note in particular the feedback loop connecting response time (represented by the sum of the activities in the shaded box on the right hand side) with concurrency. This can be defined using Little's Law [1] which relates response times and the number of in-memory processes in an open system.

Much of this dynamic behaviour is non-linear and cannot therefore be simply extrapolated. For example - the response time at each potential delay point is the sum of the busy time and the queuing time. The queuing time for a physical device is a non-linear function of its utilisation. A modelling approach that encompasses this behaviour is discussed in section 7.

3. System Global Area

In order to appreciate the importance of the memory structures contained within the SGA, the algorithms used to manage them, and their effect on physical resource consumption, we should first consider the sequence of events caused when Oracle updates a row of a table. Figure 2 below describes a simplified view.

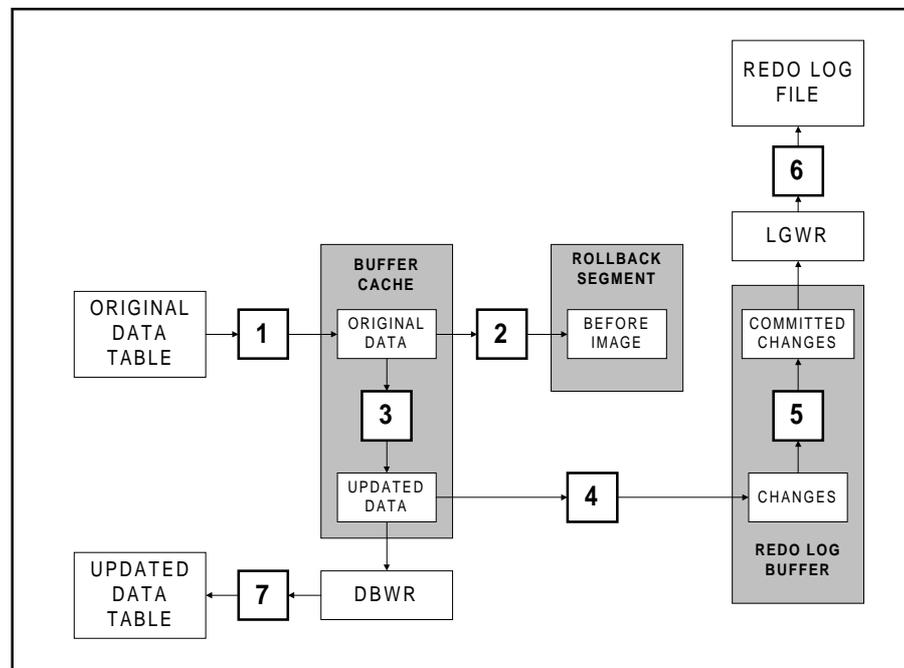


Figure 2. Life History of an Oracle Update

The events are shown in the boxes :- **2** which are numbered in sequence :

1. The row to be updated is first copied to the buffer cache in the SGA and locked.
2. A 'before image' (a copy of the original row) is written to the buffer cache for rollback. This is written to the rollback segments on disk by the database writer asynchronously.

3. The row is modified in the buffer cache.
4. The changed row is copied to the redo log buffer.
5. The update is committed and flagged in the redo log buffer.
6. The committed row is written to the redo log by the log writer (LGWR).
7. The modified row is written to disk by the database writer (DBWR)

Clearly any contention for resources relating to the buffer cache, redo or rollback mechanism will have a serious effect on throughput and response times. Any contention for the Library Cache (used to store shared SQL) and the Dictionary Cache (used to store object definitions) will also affect response times. Each of these areas is considered separately below.

3.1 Buffer Cache

The buffer cache is used to store database blocks that contain tables, indexes, clusters, rollback segments and sort data. Each buffer holds a single database block. The number of buffers is defined in the INIT.ORA parameter DB_BLOCK_BUFFERS. The important metric here is the probability that Oracle finds the block that it needs already in the cache, rather than having to get the block from disk. This is referred to as the buffer cache hit-ratio. It is calculated from the I/O data provided in the dynamic performance view V\$SYSSTAT using the formula:

$$\text{hit-ratio} = (1 - \text{physical}/\text{logical}) * 100.$$

The number of physical reads is provided directly in V\$SESSTAT. Logical reads are calculated from the sum of 'consistent gets' and 'db block gets'. Individual user session hit-ratios are calculated using the V\$SESSTAT view.

The usual recommendation is that the hit-ratio should be above 95% for OLTP and above 85% for batch applications. The interaction between the available physical memory and the working set size will have a non-linear effect on the buffer cache hit-ratio. The effect that increasing the number of buffers will have on the cache hit-ratio can be assessed by setting the INIT.ORA parameter DB_BLOCK_LRU_STATISTICS to 'TRUE'. This will place statistics in the X\$KCBCBH table which will show the hits that would have been achieved had more buffers been available. Remember that this procedure should be used on a temporary and controlled basis due to the overhead that it places on the system. If you are only able to look at one aspect of Oracle performance, the database buffer cache hit-ratio should be your choice.

A word of caution is appropriate here. Remember that, when Oracle records a 'physical read' what it means is that Oracle has issued a data request to the operating system. This may or may not result in an actual I/O to disk, depending upon the HP/UX buffer cache hit-ratio. It is possible, on certain environments, to use raw I/O to disks, in which case Oracle bypasses the UNIX buffer cache. Using raw I/O makes managing disk space much more difficult, and many systems do not support it.

3.2 Rollback

The rollback segments are used for read-consistency (enabling Oracle to avoid the need for read locks) rollback and recovery. The *key* metric here is the rollback segment get hit-ratio which is calculated from data provided in V\$ROLLSTAT using the formula:

$$\text{get hit-ratio} = (1 - \text{waits/gets}) * 100.$$

Any figure less than 99% for the get hit-ratio is indicative of rollback contention - possible causes are too few rollback segments or rollback segments that are too small in proportion to the volume of updates.

3.3 Redo Log Buffers

The Redo Log buffers are used to hold changed data prior to writing them to the on-line Redo Logs. An update-intensive application can cause contention for access to the redo log buffer and this can be diagnosed by using data from the V\$LATCH view. V\$LATCH gives the number of gets, misses, immediate gets and immediate misses. If the ratio of misses to gets, or of immediate misses to immediate gets is greater than 1%, then the Redo Log buffer size should be increased. One can also monitor time spent waiting for Redo Log space using V\$WAITSTAT. This provides direct input to response time analysis.

3.4 Shared Buffer Pool

The Shared Buffer Pool contains two vitally important memory structures - the Library Cache and the Dictionary Cache. A single INIT.ORA parameter (SHARED_POOL_SIZE) controls the total size of the Shared Buffer Pool including both the Library Cache and the Dictionary Cache. The main job of the Library Cache is to hold shared SQL statements that have been parsed. Clearly a significant performance gain can be obtained if Oracle can re-use an SQL statement that has already been parsed.

The key metrics are available from the V\$LIBRARYCACHE view:

- number of gets
- the Library Cache get hit-ratio
- number of PINs
- the PIN hit-ratio
- number of reloads

The number of PINS represents executions of an item in the library cache. Both the Library Cache get hit-ratio and the PIN hit-ratio should be greater than 90%. The number of reloads gives the number of times that a cache entry is missed due to the fact that it has been 'aged out' through lack of space. The number of reloads should never be more than 1% of the number of PINs. If it is, then the SHARED_POOL_SIZE parameter should be increased.

The Dictionary Cache is used to hold definitions of dictionary objects in memory. Dictionary Cache gets and misses are reported in the V\$ROWCACHE view. The hit-ratio (once the database has been loaded and used for a little while) should be better than 90%.

3.5 Case Study

As recommended above, one of the first metrics that we looked at was the buffer cache hit-ratio, the table below shows a sample from the peak period (1100 - 1230).

Start Time	Hit-ratio
11:00:02	99.2
11:15:02	98.6
11:30:01	97.3
11:45:02	96.9
12:00:00	97.3
12:15:01	95.8
12:30:01	97.5

This shows a very high hit-ratio, well in excess of the usual targets. We also looked at the individual session buffer cache hit-ratios which, for the users that used the majority of the resources, showed average cache hit-ratios in excess of 98%. The other hit-ratios referred to earlier in this section were also checked and found to be within acceptable limits. Clearly from an Oracle point of view, contention for SGA resources was not the source of the problem.

We had been informed that the live Oracle instance was the major user of CPU resources on the system. This we checked by correlating the CPU consumption reported by the HP/UX *sar* utility, with the CPU consumption reported by Oracle. This is shown graphically in figure 3.

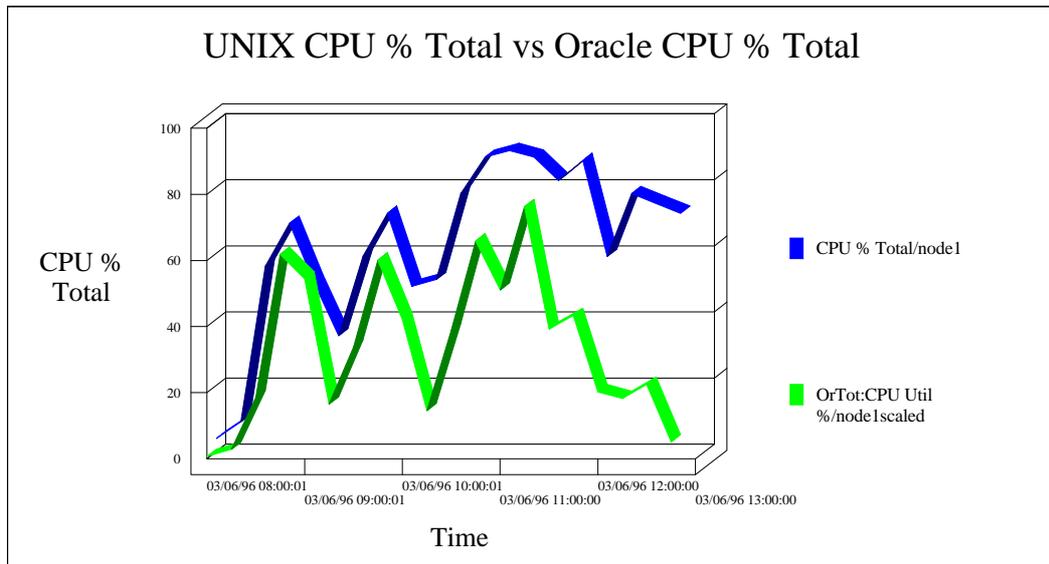


Figure 3. HP/UX CPU vs. Oracle Live CPU

The x-axis on this graph covers the period 0800 - 1300. The CPU usage reported for the Oracle 'Live' instance is a significant proportion of the total CPU for the majority of this period. Note that peak (O/S) CPU is over 80% at the morning peak time of 1100. CPU utilisations at this level will cause significant CPU queuing with a consequent effect on response times.

The next area that we looked at was the way in which HP/UX memory resources were being used. The graph in figure 4 shows a correlation between the amount of free HP/UX memory and one of the key paging metrics - the number of physical page-outs per second.

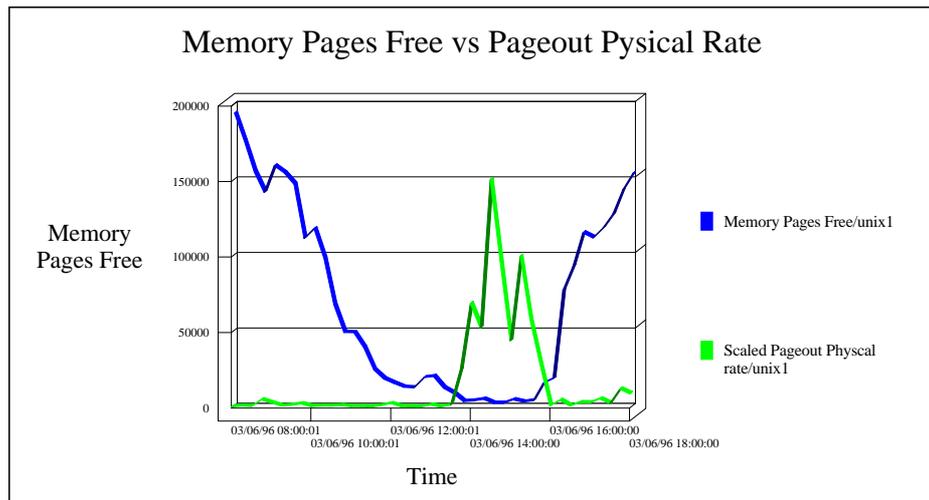


Figure 4. HP/UX Free Memory vs. Pageout Physical Rate

This dramatic picture shows the effect of increasing main memory pressure. As the amount of free memory falls during the first half of the day, the pageout rate stays very low - until free memory becomes critical, at which point physical pageouts start to increase rapidly. As demand for HP/UX memory decreases later in the day the pageout rate once again drops back to very low levels. This heavy I/O load, and the CPU required to service it, will damage application response times during the critical peak period.

Oracle's 'rule-of-thumb' for HP/UX systems (and this should be used with caution since different application mixes will have different requirements) is that the total SGA size for all instances should not be more than one third of the total physical memory. This is necessary to provide enough space for HP/UX to manage the whole environment.

In this case it was possible to reduce the SGA size, by reducing the number of database block buffers, without any significant effect on database buffer cache hit-ratios. This, in turn, reduced the pressure on UP/UX memory resources and virtually eliminated the paging activity shown above.

4. Disk Storage

This section concentrates on the physical organisation of data and the ways in which contention can cause performance problems. The advice provided by Oracle on tablespace allocation should be followed closely. It is summarised here for completeness; each site should check that the Oracle recommendations for their platform have been followed.

4.1 Tablespace Configuration

The tablespace is the largest logical unit of space in an Oracle database. All databases should have at least six tablespaces as follows, although many more may be required in order to separate data of different types and with different uses.

- SYSTEM
- RBS
- TEMP
- USERS
- *xxx_DATA*
- *xxx_INDEX*

The SYSTEM tablespace should contain only data dictionary objects. A separate tablespace (RBS) should be allocated for rollback segments. A temporary tablespace should be used for sorts etc. The USERS tablespace is for those (few) users who need to create objects.

4.2 Database Files

Data on file usage is obtained from the V\$FILESTAT view. The primary objective should be to spread the Oracle files across the available physical devices in order to distribute the I/O evenly, and to prevent any single device from becoming a performance bottleneck. In particular it is worth checking for any heavy I/O activity to the SYSTEM tablespace files, which could cause significant response time problems. This I/O could be caused by the Shared Buffer Pool being too small, or by data and/or sort segments being written to the SYSTEM tablespace.

Although it is possible to stripe files across physical disks by hand, many UNIX implementations support striping directly and this should be used if possible. The Oracle literature itself states that striping by hand is labour-intensive and offers few advantages. Check, using the HP/UX *sar* utility, that striping is in effect and that disk I/O load is evenly distributed.

4.3 Database Blocks

One of the goals of any tuning exercise is to minimise the number of database blocks that are read in order to retrieve the required data. This can be achieved in a number of ways:

- using a larger block size
- packing rows as closely as possible
- preventing row migration
- tuning SQL statements

It may be possible to use a larger block size. This will be constrained by the HP/UX release that you are using. Note that the database will need to be re-built if you change the block size (which is specified in the INIT.ORA parameter DB_BLOCK_SIZE). Blocks can be monitored using the ANALYZE command which populates the DBA_TABLE view.

4.4 Fragmentation and Contention

Data fragmentation occurs in a number of different ways:

- internal fragmentation - caused by deletions or updates within a block
- external fragmentation - where a table is stored on multiple (non-contiguous) extents
- row migration - where an updated row will no longer fit in its original block, the entire row is 'migrated' to a new block and pointed to from the original block
- row chaining - where a row is too large to fit in a single block

All of these cause additional I/O in order to satisfy each logical request. The DBA_TABLES view can be used to detect this behaviour.

Another possible cause for contention is freelist processing. Oracle maintains at least one freelist in each segment header. Processes that are looking for blocks to insert into, search the freelist. In a busy system where there is considerable insert and delete activity, processes may contend for segment freelists. Any freelist waits are logged in V\$WAITSTAT.

4.5 Case Study

An analysis of traffic by Oracle file for the application system, during the peak period, provided the following results:

File Name	Reads	Writes
houtabLIVE1.dbf	1436141	4316
houidxLIVE3.dbf	713839	6361
houidxLIVE1.dbf	197294	1155
houtabLIVE3.dbf	195373	1643
rbsLIVE1.dbf	158966	7200
houtabLIVE2.dbf	84086	249
sysLIVE1.dbf	25470	1184
houidxLIVE2.dbf	12488	278
toolsLIVE1.dbf	12383	5

This list is sorted on the total number of reads for each file and shows that I/O activity is concentrated on a small number of files. More than half the total number of reads was to the file 'houtabLIVE1.dbf'. A check on the physical I/O by HP/UX disk revealed the following picture:

Physical Disk	Utilisation
n03sd01b	14.6
n02sd001	10.2
n01sd01a	7.2
n02sd029	4.5
n03sd01a	4.3
n02sd03b	2.9
n02sd002	2.8
v00sd006	2.7

This table is sorted on the total utilisation of each device and does not show the even distribution of I/O that one would normally expect on a striped filesystem. A check quickly revealed those *none* of the Oracle database files were striped, which resulted in the unequal distribution shown above.

5. SQL and Database Navigation

5.1 SQL Standards

As already mentioned in section 3.4 above, significant performance gains can be obtained by exploiting Oracle's ability to re-use parsed SQL statements. For SQL to be re-used it must be identical. This is much more likely to be the case if all SQL code is written to an enforced set of standards, and bind variables are used instead of literals.

5.2 Tracing SQL

The Oracle `SQL_TRACE` function can be used to analyse SQL statements. `SQL_TRACE` can be enabled for the entire instance (this is not practicable due to the performance hit and volume of data that would be produced) or for specific sessions. The output is written to a trace file that must be processed using another Oracle utility - `TKPROF`. `TKPROF` produces resource consumption and performance details for each SQL statement and (optionally) an `EXPLAIN PLAN` which shows the way in which the optimiser will process the SQL and the optimiser mode used. The SQL optimiser examines each SQL statement and chooses an optimal execution plan. Oracle 7 gives the choice of two optimisers - the old rule-based optimiser and the newer, more intelligent cost-based optimiser. The optimiser mode can be set at instance level, session level or statement level (using `HINTS`).

5.3 Database Navigation

When reading rows from a short table, or when the majority of rows are required from a long table, a table scan is usually the most efficient method. When only a few rows are required from a long table, it is usually more efficient to use an indexed read. Clearly this is only possible when an index has been defined on the table using the required key. The output from `TKPROF` and `EXPLAIN PLAN` can be used to identify missing (or mis-used) table indexes by showing the number of rows scanned, number of rows retrieved, and the access method used.

5.4 Case Study

An analysis of the CPU consumption of Oracle sessions for the peak period gave:

SID	Serial	Oracle User	CPU seconds
103	89	OPSS\$WEST01	2584.41
80	173	OPSS\$MBARN01	2583.93
53	579	OPSS\$WEST01	1224.62
34	487	OPSS\$WEST01	1215.47
39	33	OPSS\$WEST01	1179.52
32	137	OPSS\$WEST01	1129.08
107	385	OPSS\$TSALM01	938.11
109	297	HOUQUERY	912.87
98	81	OPSS\$NPASC01	907.28
84	563	OPSS\$CDRAK01	393.84
105	49	OPSS\$CDRAK01	270.79
119	411	OPSS\$DCANN01	263.55

This list is sorted on the total CPU consumption of each session. Note the predominance of user `SWEST01`. Further analysis showed significant numbers of long table scans for this user. `SQL_TRACE` was then enabled for all active sessions where user name was `SWEST01`, followed by `TKPROF` including the explain plan option. This showed that, of the 149 SQL statements analysed, just 14 (9%) were responsible for 97% of the CPU time. Each one of these 14 included a full table scan of the same table. Each time 11,812 rows were scanned and less than 20 rows were returned. A simple addition of an index to this table reduced total I/O by more than 20%.

6. Contention for Logical Resources

6.1 Locks

Oracle V7 uses automatic row level locking to maintain the consistency of the database when it is updated. Locks are only applied to database updates, and in the majority of on-line systems locking activity will probably affect only a small number of users or transactions. A lock is set by a process when it wishes to change a row in the database. If a concurrent process also wishes to change the same row while the first process is implementing its update, then the second process has to queue to obtain possession of the lock. Oracle does not enforce read locks, although applications can lock data if they wish. Where required, the rollback segments are used to provide read-consistency.

The probability of lock contention affecting performance depends on the locality of updates, which is a function of the application design and the transaction mix at any time. Details on locking activity are provided by the V\$LOCK view although any lock contention will only be reported in real time. The Oracle *utillockt.sql* script can be used to provide information on lock 'waiters' and the lock modes requested and held.

6.2 Latches

A latch is an internal mechanism that is used by Oracle to ensure the consistency of shared data structures. Each major data structure is protected by one or more latches. Latches should only be held for very brief (micro-second) periods. Any latch waits are indicative of very serious problems. A complete list of latches is provided in V\$LATCHNAME. V\$LATCHHOLDER gives details of all processes holding latches and V\$LATCH provides overall latch summary statistics.

7. Modelling Oracle Systems

7.1 The Need for Modelling

Much of the dynamic behaviour of an Oracle system is non-linear and cannot therefore be simply extrapolated. The Oracle environment itself is subject to the constraints placed upon it by its operating system environment. There are also a number of significant feedback loops which will affect performance as identified in section 2 above.

Analytical modelling, based on queuing network theory, has been used for some time now to model the performance of computer systems, including the non-linearities that contention for resources causes[2]. Such techniques have recently been extended to encompass the behaviour of application environments built using RDBMSs such as Oracle[3]. It is impossible to characterise their behaviour without using a model - observations are meaningless unless they can be interpreted, and related by reference to some conceptual framework.

The key contribution that using such a model makes is its ability to translate an end-user workload into the resources (both physical and logical) that the workload will both use and contend for, and derive the response time that will be provided. Once built and validated from actual baseline observations, the model can be used to explore the consequences of changes in workload and/or hardware configuration.

All analytical models represent service centres (resources that provide a service to the workload and/or may contribute a delay) and workloads. Each workload is described by its requirement for service at each of the service centres. Established modelling methods are adequate to model contention for

physical resources and the consequent effect on response times. Much of the recent work [4] has been aimed at modelling contention for logical resources such as locks and latches.

A full explanation of the modelling techniques required to represent the behaviour of locking mechanisms is outside the scope of this paper but, for example, a lock can be considered as a service centre. Clearly when a workload does not experience lock contention, its delay time at the lock service centre (to acquire the lock that it needs) is very short. If it encounters a lock it will wait until the lock holder releases it.

Using a combination of the data available from UP/UX, and the data available from Oracle (primarily from the V\$ views) it is possible to construct quite detailed models of the behaviour of Oracle applications. One should always remember however, that the model should only be sufficiently detailed to answer the specific planning questions. A model is, by definition, a *simplification* of reality, built for a specific purpose.

7.2 Case Study

Using the data from UP/UX and from Oracle, we built a model of the system referred to earlier. The main decisions which had to be made were the choice of the modelling ‘window’ (the period of time whose workload the model is required to represent) and the way in which the workload should be broken down into workload components.

The modelling window chosen was the peak morning hour, 1100 - 1200. This was chosen on the basis of the overall CPU consumption and I/O rates reported for both the Oracle instance and from HP/UX. The primary question that the modelling exercise was to address was “Can the system cope with the expected increase in the number of concurrent Oracle maintenance sessions and, if not, what actions are required to make sure that it can?” The workload was therefore broken down into components that represented:

- the Oracle maintenance sessions in the ‘LIVE’ instance
- the remainder of the work in the ‘LIVE’ instance
- the work from other Oracle instances
- the work generated by HP/UX itself in order to manage the environment

The initial or ‘baseline’ model was calibrated against the observations of device utilisation and queue lengths. It was not possible to perform a response time calibration due to the absence of any meaningful response time measurements either from HP/UX or from Oracle. This does not invalidate the model that is produced. The model will define *relative* changes in response times, rather than absolute values, for the workload components that were selected.

A projection (or ‘scenario’) model was then built to show the way in which maintenance response times would change as the overall loading increased. The non-linear effect referred to earlier is clearly seen in the response time scenario report shown below.

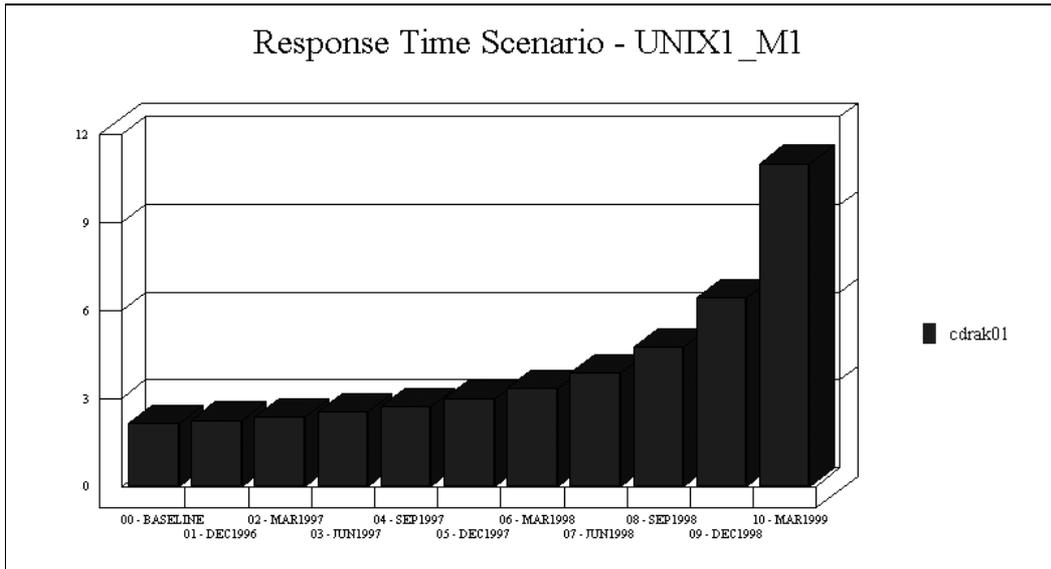


Figure 5. Response Time Scenario Report - 1

The first vertical bar represents the current (baseline) response time, and each subsequent bar represents the addition of five more concurrent maintenance sessions. A second scenario model was then built incorporating two changes from the model shown above:

- (a) The CPU was upgraded from an 4-way to a 8-way at projection point 5
- (b) An index was added (to the table referred to in section 5.4 above) at projection point 7

The effect on maintenance response times is shown in figure 6 below:

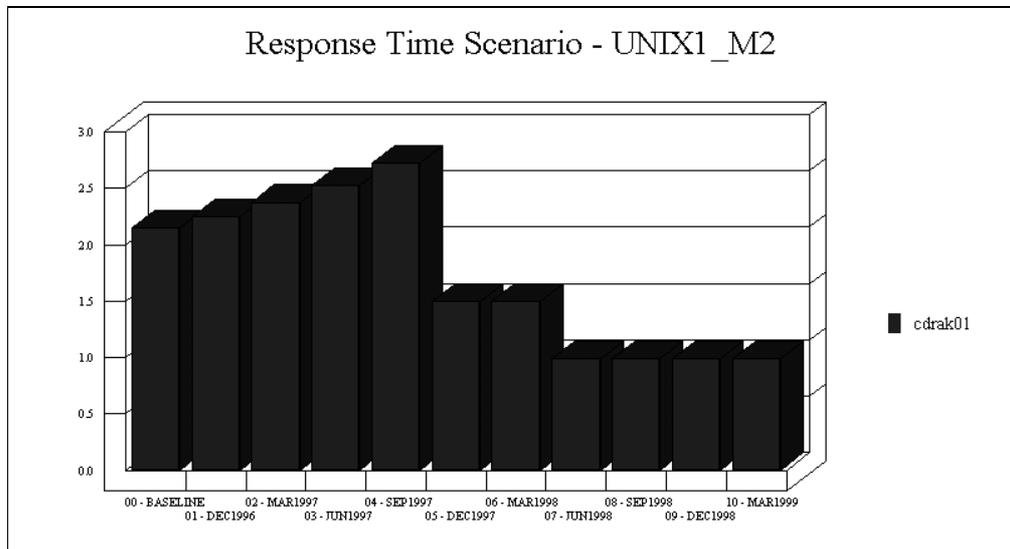


Figure 6. Response Time Scenario Report - 2

The advantage of this approach is that a large number of alternative workload and configuration combinations can be assessed, *from the point of view of their impact on end-user response times.*

Many of the options that are available for changing Oracle and HP/UX configuration parameters are only effective at startup time. The ability to assess the likely impact of those changes in advance saves both time and money, and is a good deal safer than the experimental approach!

8. Summary & Conclusions

The main message of this paper is that we should not allow ourselves to be put off by the apparent complexities of Oracle system behaviour. By selecting and concentrating on a small number of key metrics, significant performance improvements can often be gained.

The paper includes definitions of the sources of the important metrics. These are accessible through standard SQL or HP/UX commands. One of the main problems with the raw metrics is that they are often snapshots at a single instant - and the important thing for characterising system behaviour is not the absolute values, but the way in which those values change over time.

Software tools are now available to assist in the process of capturing and storing this data. Such tools also speed up the process of identifying patterns, trends and interrelationships between the observations.

Modelling techniques are available which will allow us to predict the future behaviour and performance of Oracle systems. This will enable accurate forecasts to be made of the resource implications of changing workloads - enabling systems to be managed in a proactive, rather than a reactive way.

REFERENCES

1. A proof of the queuing formula $L=\lambda W$, J.C. Little, Operations Research Vol. 9, 1961, pp. 383-387.
2. Performance Management of Client-server Systems, Adam Grummitt, CMG Proceedings, 1994
3. Capacity Planning in Client/Server Systems, T. Foxon, M. Garth and P. Harrison, Journal of Distributed Systems Engineering, Vol. 3, 1996, pp 32 -38.
4. Capacity Planning for Parallel IT Systems, M. Garth, Computer Bulletin, Vol 8, part 5, November 1996, pp 16 -18.