A queueing network model of Oracle Parallel Server

Uli Harder *and Peter Harrison †

September 18, 2003

Abstract

An approach to modelling an Oracle database is presented for both single instance and parallel server versions. A modified queueing network is used which is extended to include traditional transaction locks which achieve serialisation and the new PCM locks of Oracle which ensure coherency in the distributed database buffers. The latter are not enqueue locks and so require different techniques. Both locks require an iterative solution to a fixed point model. The numerical results produced by the model can be validated largely using data maintained in Oracle tables and the distributed lock manager, and we identify the relevant entries.

1 Introduction

Concurrency in databases like Oracle is usually managed by the use transaction locks and the contention arising from their use has been modelled by iterative methods typified by [1]. In the case of a parallel architecture where several instances use a shared database, the buffer areas or caches in the instances have to be kept in a coherent state, i.e. every instance must see the same value locally in its cache for a particular block of data. Again locks are used to facilitate this. For a good review of the relevant parallel architectures, see [6]. A way to model such an architecture is proposed and combined with expanded earlier work on transaction locks to model Oracle parallel server (OPS). Also, the important differences between OPS 7.3 and OPS 8.0 are highlighted.

The aim throughout the paper is to present a non-trivial model whose input parameters are easily obtained from the database without making any changes to its operation. The obvious choices for sources of information are the V$ or, in the case of OPS, GV$, tables. A crucial aim is to forecast when lock contention, be it for coherency or concurrency, becomes a non-negligible factor of the response time. There is expected to be a distinct saturation point for the number of concurrent transactions executing, as is observed in many systems with shared resources such as shared virtual memory systems (with 'thrashing') and Ethernet [2]. Given the number of transactions, number of read processes, number of lockable data per tablespace, number of locks actually acquired per tablespace, likelihood to acquire a lock in a particular tablespace and CPU and

*Uli.Harder@metron.co.uk, Metron Technology Ltd., Taunton
†pgh@doc.ic.ac.uk Imperial College, London

I/O service times, the model predicts the average queue length and the average time to wait for a lock per tablespace. It is therefore possible to examine the effects of different access patterns on response time, for example. It seems conceivable that the proposed model could be adapted to other database systems, though it may be necessary to make significant modifications.

The paper is organised as follows. First the working of Oracle Parallel Server is explained in section 2. Then a short introduction is given to transaction locks in section 3. PCM locks are then explained in 4 and a model for a parallel database is developed in 5. The paper concludes in section 6, suggesting immediate and longer term research directions.

2 Oracle Parallel Server

Instead of modelling the most general parallel server architecture Oracle parallel server [3, 5] was chosen as an example. How easily the results will be generalised to other architectures (and there are many) remains to be seen. Unfortunately the description of a system as a parallel server or cluster is completely insufficient for modelling purposes. Rather than introducing all available parallel architectures only the OPS will be described. OPS, especially version 8.0, can run on almost any hardware. However the most likely candidates are shared disk or shared nothing architectures. This would include a massively parallel processor (MPP) and a cluster, as shared disk tightly or loosely coupled systems. Also, there would be a shared nothing MPP and a shared nothing loosely coupled cluster, like Microsoft's NT cluster or IBM's SP2.

In version 7.3, OPS required a distributed lock manager (DLM) which would come with the chosen platform. The latest version 8.0, however, comes with an integrated distributed lock manager (IDLM). This development makes it easier to model OPS as the IDLM basically behaves like the DLM, but OPS actually has additional 'V$' tables with metrics about the IDLM. The (I)DLM will normally communicate with the processing nodes via a high speed interconnect, which is simply a dedicated network. On NUMA and SMP systems, OPS is not the best choice because of the unnecessary overhead created by the (I)DLM. On these machines, using the Oracle Parallel Query option should bring more benefits.

The OPS is a shared disk parallel application—not to be confused with a shared disk architecture. Typically, it consists of one or more instances of Oracle servers. These servers share a database, i.e. there is only one physical set of database files, but many instances that access the data simultaneously. Also, instances can access each other's redo logfiles for fail-over purposes. Typically these instances would run on nodes of a cluster. However, the instances can reside on one single computer or, at the other extreme, the nodes can be single computers on a network which only behave as a cluster once the OPS is installed—as in the case of NT, where Microsoft Cluster Service (MSCS) is not enough to run OPS.

Transaction locks exist in this arrangement to achieve synchronisation and serialisation in the same way that they do in the single instance case. The shared disk architecture has the consequence that the only way to exchange information between nodes is via the database files. However, the nodes (instances) can exchange information about the state of data via the (I)DLM, which is a piece

of the cluster OS or the OPS. Internally (with respect to OPS) the messages are exchanged using parallel cache management (PCM) locks. These PCM locks are the real trouble-makers and make modelling OPS different from single instance Oracle. PCM locks behave differently from transaction locks as they are not enqueue locks so that requesting instances do not have to wait for releases of PCM locks until the end of an operation. A request is satisfied almost immediately, and the current holder of a lock loses it when it is requested by a different *entity*. As the last sentence indicates by choosing the word entity there is another significant difference: PCM locks are held by instances not transactions. Exclusive PCM locks can be shared within an instance.

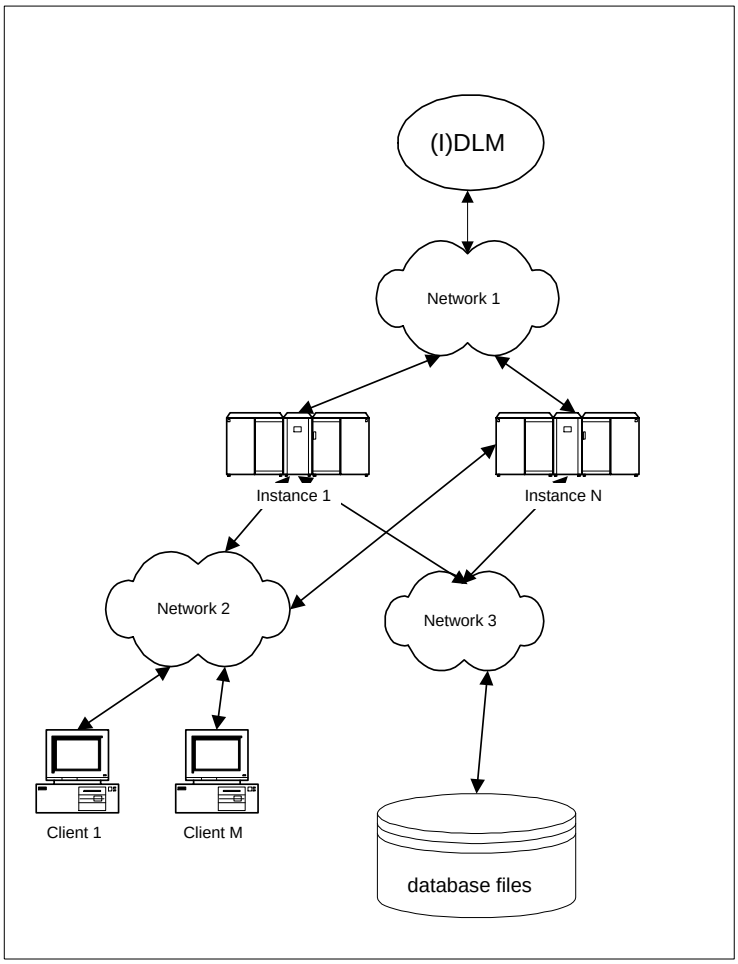

Figure 1: Sketch of an Oracle Parallel Server arrangement

For read transactions PCM locks can be shared between instances. Updates make it necessary for instances to have an exclusive lock. It may seem surprising at first that an instance can lose its exclusive PCM lock before all its transactions have committed. But assume there are 4 transactions on 2 different instances. T11 and T12 live on instance 1, and T21 and T22 live on instance 2. The write

transactions T11 and T12 start first and share the PCM lock PL. Before T11 and T12 commit, T21 starts and requests read share PCM lock PL. As there is no waiting for PCM locks, T21 gets PL immediately and because neither of the other transactions have committed, it reads in its data from their rollback segment. Now T12 signals it wants to commit. Hence instance 1 has to re-acquire the lock PL in exclusive mode. T12 writes out its new data. Now T22 starts an update on data covered by the PL lock but not the transaction lock of T11. The first thing to happen is that instance 1 writes out dirty data caused by the commit of T12 to the database. Then instance 2 acquires the lock PL and the transaction T22 starts by writing data to the rollback segment. When T11 signals it wants to commits it acquires the lock PL and then commits. When instance 2 re-acquires the lock it may cause instance 1 to write out dirty data if it intends to read that data. This example shows that in OPS there are two layers of locking. One layer is given by the familiar transaction locks which guarantee concurrency control. The other, new, layer makes sure the SGA of the all instances are coherent. Those two layers are completely independent of each other. As the diagram in figure 1 shows it is possible to view the problem as that of multiple networks. However as the DLM is not independent of the nodes, the networks influence each other.

3 Transaction locks

3.1 Operation of transaction locks

We divide the I/O of the entire Oracle instance by tablespaces. Exceptions are the logfiles, about which there is little information but which do not have a large impact on performance. Similarly, there is no detailed information about writes to the rollback segments, because they are contained in the tablespaces. The I/O of tablespaces can be calculated from V$TABLESPACE. For the logfiles, V$SYSSTAT combined with V$STATNAME gives the number of blocks written and write time for the entire instance.

We divide the processes running in an Oracle instance into three categories:

1. Updates and Deletes
   (a) Parse and optimize the SQL statement
   (b) Read data from cache or file, possibly wait for incompatible locks in the process.
   (c) Write old values to the rollback segment
   (d) Compute new values
   (e) Replace or delete values in the database file
   (f) write old value to the redo log file
   (g) proceed with 1a) or 1h)
   (h) Commit: write out entries in the redo log buffer to file
   (i) Release all locks
2. Inserts, which behave similar to updates.
3. Reads (and serializable reads)

(a) SQL statement is parsed and optimized

(b) Selected data is read from file/buffer

(c) Some calculation for the presentation may be performed

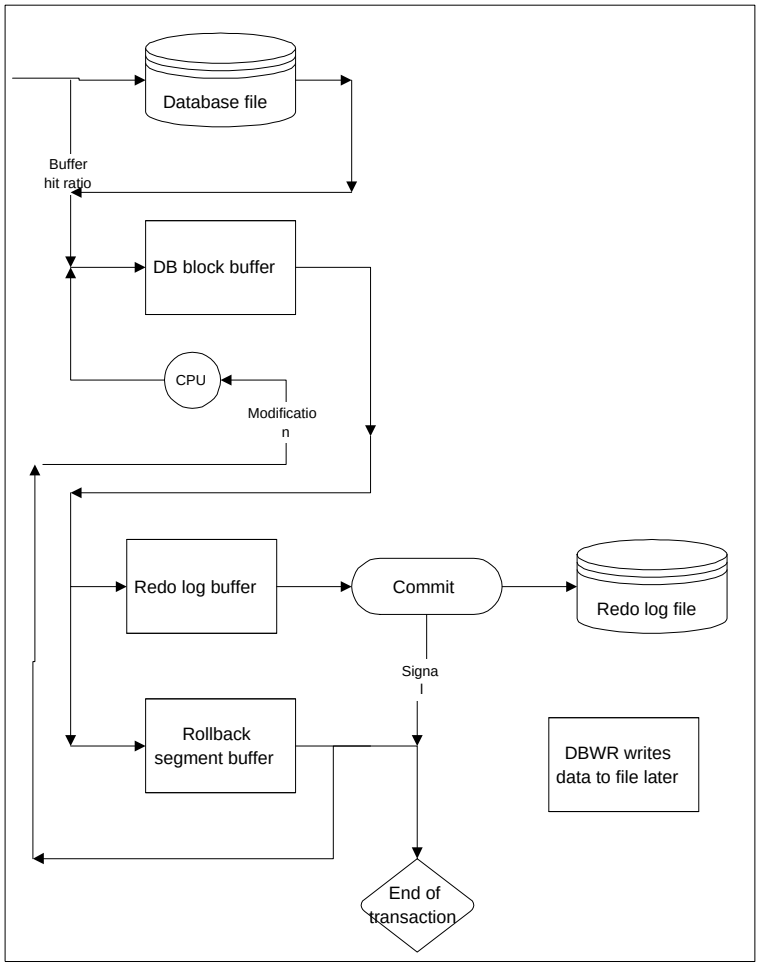

Figure 2: Oracle Update

Both of the distinct logical cases can be modelled using a multi-class model. The different process classes can be represented either by different job classes or by appropriate routing probabilities. Even for this rather detailed model one has to make simplifying assumptions about the way locks are acquired. In reality the data would be read row by row and interrupted at any time by access conflicts.

3.2 A single class model - Thomasian's approach

For computational purposes it may be simpler to use a single class model where the different classes are accommodated by routing probabilities.

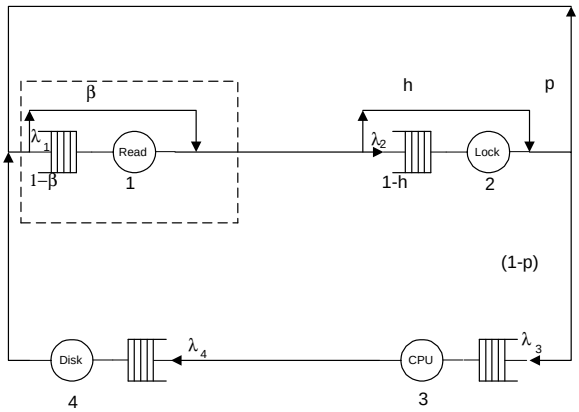

Figure 3: Single class model

For simplicity the service stations Read, Lock, Disk and CPU are numbered 1, 2, 3 and 4 respectively. This model can be used to describe Updates and Deletes as follows:

1. A transaction tries to read data from the DB. Depending on the buffer hit ratio, this may be read from file or buffer. (1)

2. The transaction may find that some of the data is locked. This means it has to wait for the release of the locks. The service time will depend on the overall response time of the system. The probability of a lock conflict depends on the number of locks held by transactions. (2)

3. Then the transaction may decide to either read more data or do some calculation at the CPU. (1) or (4) with probability p and $1-p$ respectively.

4. Calculations at the CPU are followed by disk activity to write the data out to the data and log files. Partly, this will be written to buffers. (4)

The probability p determines the number of locks acquired by each transaction on average:

$$L = (1-p)\sum_i ip^i = \frac{p}{1-p}. \tag{1}$$

All service stations have a fixed service rate, which is not queue length dependent. The service time for the Lock server is determined iteratively. In the first step the service time is set to zero. In the following steps it will be proportional to the overall response time of the previous step. The probability h determines the likelihood not to find a lock conflict; see [1]. Notice that a real lock has zero service time if there is one or less customers attached to it. In our model, the service station 3 in combination with the probability h models exactly that. If there is no conflict, the process carries on without delay (probability h). If there is a conflict (probability $1-h$) the process has to wait at a lock. Thus,

$$h = 1 - L\frac{M-1}{2N} \tag{2}$$

Here N is the average number of lockable items and M is the total number of customers. The probability to bypass the reading of data from the disk is β: this is simply the buffer hit ratio.

Now one needs to work out the arrival rates λ_i for a single class model. The arrival rates have to satisfy the traffic equations $\lambda_i = \sum_{j=1}^{M} \lambda_i q_{ij}$, where q_{ij} is the probability for a job to pass from service centre i to service centre j on completion of service at centre i. The set of equations is

$$\begin{aligned}
\lambda_1 &= (\lambda_2 p + \lambda_4)(1-\beta) + \lambda_1 hp(1-\beta) && (3)\\
\lambda_2 &= \lambda_1(1-h) + \lambda_2 p\beta(1-h) + \lambda_4\beta(1-h) && (4)\\
\lambda_3 &= \lambda_1 h(1-p) + \lambda_2(1-p) + \lambda_4\beta h(1-p) && (5)\\
\lambda_4 &= \lambda_3 && (6)
\end{aligned}$$

Given λ_4 the arrival rate λ_3 is immediate and the remaining two arrival rates also follow quickly as

$$\begin{aligned}
\lambda_1 &= \lambda_4 \frac{1-\beta}{1-p} \frac{1}{(1-p\beta h(1-h))^2} && (7)\\
\lambda_2 &= \lambda_4 \frac{1-h}{1-p} \frac{1}{1-p\beta h(1-h)} && (8)
\end{aligned}$$

The arrival rates can then be used to determine the visit ratios $\nu_i = \frac{\lambda_i}{T}$, where T is the throughput of the network. We can follow the recipe in the book [4] (p. 239) and introduce an extra arc with node numbered 0 between the CPU and the Disk write server. The new traffic equation is $\nu_0 = \nu_{CPU} q_{CPU,DW}$ and because $\nu_0 = 1$, it follows that $\nu_{CPU} = \nu_3 = 1$. The remaining visit ratios can be deduced from the traffic equations as

$$\begin{aligned}
\nu_1 &= \frac{1-\beta}{1-p} \frac{1}{(1-p\beta h(1-h))^2} && (9)\\
\nu_2 &= \frac{1-h}{1-p} \frac{1}{1-p\beta h(1-h)} && (10)\\
\nu_4 &= 1 && (11)
\end{aligned}$$

This model is now solved using standard methods. As before the overall mean response time is determined by iteration:

1. Set $\frac{1}{\mu_2} = 0$ and compute $R = W(K)$, the response time;
2. Set $\frac{1}{\mu_2} = hLS$, with $S = fR$, and compute R again by solving the queueing network model;
3. if $|R_{old} - R|/R < \epsilon$ for some prespecified precision ϵ, then stop. If the plot of R against S is steeper than $1/f$ then the iteration is diverging, so stop as well. Otherwise repeat from step 2.

This model has some obvious limitations; on the number locks and locked data etc. This is due to the fact that h has to be in the range $0 < h < 1$.

4 (I)DLM AND PCM LOCKS

Before a model can be presented, a short introduction to the world of (I)DLM and PCM locks should be given.

4.1 (I)DLM LOCKS

In versions of OPS before 8.0, the DLM was a platform dependent piece of software that managed the access to resources within the cluster. It controlled all logical operations and data structures. However, it had no direct interaction with the database as the database mainly used to manage parallel cache management (PCM) locks. With the advent of version 8.0, OPS has an integrated DLM (IDLM) which behaves very much like the original DLMs.

The DLM holds locks for every single resource in the cluster and does not necessarily have to be hosted on one particular node but can be distributed. This is advantageous for fail-over and load balancing; the latter especially with respect to the amount of memory used by the DLM. (Each lock on a resource will consume memory.) When a process requests a lock it is put in a queue. Requested locks are converted (to the required mode) and granted by the DLM. The communication with the instances uses asynchronous traps (AST).

The DLM works as follows: Assume there are 2 processes A and B competing for the same lock which is currently held by B.

1. Process A contacts the DLM with a lock request
2. DLM sends BAST to B
3. B releases the lock or sets the lock to NULL
4. AAST is sent to A
5. Lock is granted to A
6. A is put into the granted queue.

There are a number of different states for DLM locks, but of more importance are the PCM locks so no further discussion is given here.

4.2 PCM LOCKS

PCM locks are owned by instances of the parallel server and are not enqueue locks in that a shared mode lock is not disowned by its owner if another instance requests a compatible mode. To be more precise, PCM locks are owned by the LckN background processes of each instance, which communicate the lock requests between instances and the (I)DLM. PCM locks ensure cache coherency of the buffers in the System Global Areas (SGAs) in the instances. They are allocated to datafiles, or more precisely, to sets of data blocks in a datafile. The way locks are allocated to blocks is determined by a number of parameters in the initialisation file.

There are two types of PCM lock: *hashed* and *fine grain*. Hashed locks are pre-allocated to sets of datablocks at the startup of the database and statically hashed. The first instance to start-up creates a DLM resource and a NULL lock. The second instance to refer to the resource has to wait until the lock is available, then conversion to the appropriate mode is done. The lock is only released on demand. Fine grain locks, on the other hand, are allocated on demand, and released when they are no longer needed.

The *lock element* (LE) is an Oracle data structure that represents any DLM lock. For hashed locks the LE is fixed whereas the LE for fine grain locks

changes. The LE contains the data block address (DBA) and the class of the block. The table FILE_LOCK gives information about the types of locks used in a datafile and the way they are allocated. Whilst blocks are in the buffer, the V$BH table contains information about their status, which can either be shared (SCUR or S), exclusive (XCUR or X) or null (CR or N). When an exclusive lock is requested, all other locks on this block are downgraded to N locks. There are a number of different classes of blocks which are all protected by PCM locks. Information about the type of a block is held in V$ tables. V$LOCK_ELEMENT shows the status of lock elements. For fine grain locks the states are valid, old, or free. Fine grain locking is used if the number of locks per file in GC_FILES_TO_LOCKS is set to 0.

When a PCM lock of a particular mode is requested, the request is granted immediately (there may be a delay due to the load on the DLM and network, however). Within an instance a lock can be shared between several transactions. But if another instance requires the block in an incompatible lock mode, the block has to be written to disk. This is called *pinging* a block. Obviously one way to avoid a high number of pings is to partition the data such that different instances access different parts of the database. Fine grain locks allow a 1-to-1 mapping of locks to blocks; this is called DBA locking. In this case there will be no false pinging, i.e. pinging due to requests for different blocks covered by the same lock. However, there is more (I)DLM overhead in this case as a lot of locks have to claimed, converted and released. Moreover, due to the amount of memory used by locks, there is usually an upper limit imposed on the number of locks that can exist concurrently.

5 A combined model for PCM and transaction locks

A model of Oracle Parallel Server has to include read transactions from the start as they need PCM share locks. In figure (4) a model database with two distinct tablespaces is depicted. In a simple model one would have a closed system with 4 classes. There would be two instances comprising the Oracle Parallel Server. Each of them having a fixed number of read and of write jobs, hence four classes in total. Similar to the model for transaction locks discussed before, the jobs would go round the system following the routing probabilities in the diagram. First, one could investigate a model were the probabilities separate the tablespaces for each instance. So, processes on instance 1 would use tablespace 1 exclusively and similarly tablespace 2 would be for the exclusive use of instance 1. For read processes the probability l_1 and l_2 would always be 1. For the write processes it would be determined in a similar fashion as the transaction locks. Equally the probability b allows to route read processes past the rollback. The server marked CPU is actually a different one for each instance. Also, read processes bypass the logfile server and the PCM lock server in front of it. The PCM locks server are delay servers as there ought to be no queue. Actual investigations with the Oracle Parallel Server will have to show whether this is actually the case. It is certainly imaginable that the delay will be increased if there are a lot of lock requests. P_1 and P_2 are simply the buffer hit ratio for the instances when read processes are concerned. With writes the situation is more

difficult and experimentations will have to be done to get theses parameters right. They are probably related to the routing probabilities l_i.

The model above should have a rather predictable result in that its response times should be marginally longer than a single instance database of comparable size, as the PCM lock requests add a further delay.

However, the model can be made more complicated by simply allowing for example instance 2 to read every second read from tablespace 1. This would invalidate all exclusive PCM locks held by instance 1 and therefore change the routing probabilities for the PCM lock server. One expect longer response times compared to the first model.

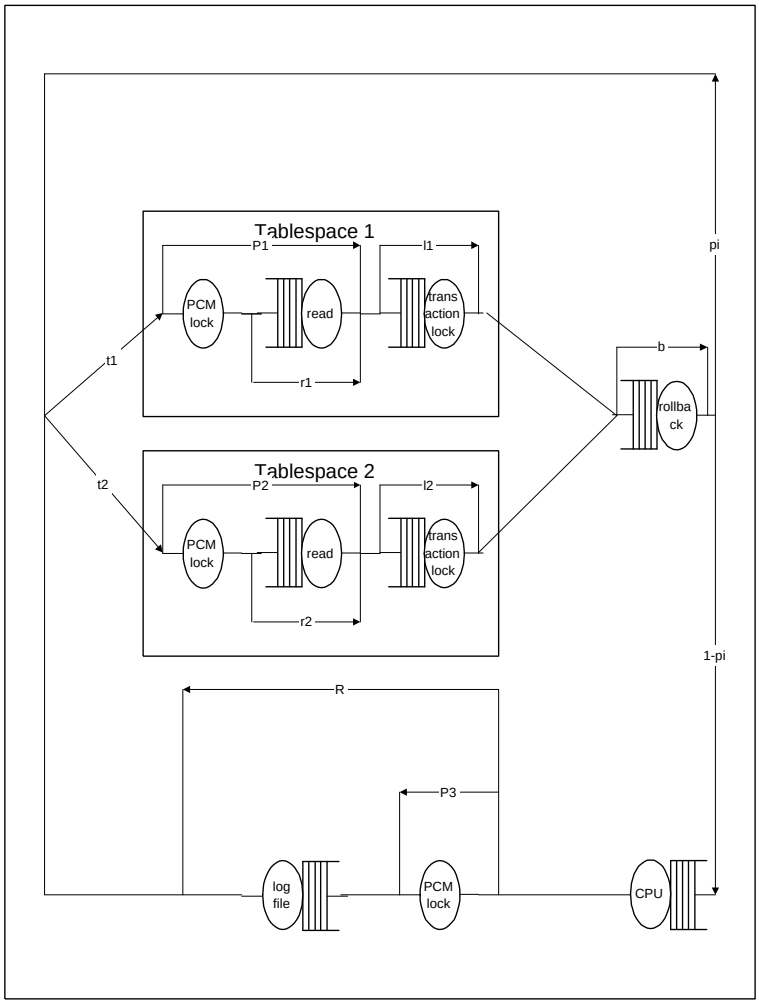

Figure 4: A multi class model of the Oracle Parallel Server with 2 instances

5.1 Model solution

As with the one-instance transaction lock model, this model needs to be solved iteratively to get the service rate of the transaction lock server. The probabilities

P_1, P_2 and P_3 for the PCM depend on different factors. The possibilities (for $P_{1,2,3}$) and their associated delays are:

1. blocks in buffer and valid lock: then no delay occurs as the instance does not have to contact the DLM

2. blocks in buffer but invalid lock: the DLM needs to be contacted, conflicting locks have to be converted, and the requested lock granted.

3. blocks not in any SGA: then the DLM lock has to be converted from N to X or S.

4. block is in buffer but its share lock needs to be updated to exclusive mode: here the DLM needs to be contacted and possibly other share locks need to be invalidated and the new lock granted.

5. there are no PCM locks left: then an existing lock is downgraded and resources are thereby freed. This, of course, causes further contention.

For P_3 only the first two possibilities are valid. The probabilities of loosing a PCM lock are given by the affinity of the data in the database to certain instances. The more data is shared, especially for updates, the more conflicts.

We implement this model in much the same way as the model with only transaction locks:

1. Run the model without taking the transaction locks and PCM locks into account. This would produce a response time per class and also, for an open model, the number of transactions per class processed per second (throughput).

2. For the transaction locks, proceed as discussed in section 3.2. For the PCM locks, we need the probabilities of a PCM clash, which depend on the database partitioning and the access pattern, i.e. software parameters, together with a sub-model to predict the delay incurred by each type of clash—see the next subsection.

3. Iterate 2, updating the model's parameters using the new predicted mean response times and throughputs.

5.2 PCM sub-model

Though the PCM locks are held by instances, the conflicts are still caused by transactions and the associated delays involve the various operations listed in the previous section. These delays require a sub-model to be constructed which can be used to parameterise a flow equivalent server node on the main model or else incorporated directly. The sub-model will be a standard queueing network with parameters depending on the state of the main model. The component delays consist of communications between the DLM and instances, lock conversions and transmissions of remote blocks. The time it takes to convert PCM locks can be measured. Contention for the DLM and for disk accesses can be modelled by conventional queues but the communication delays will normally need a special node to represent the interconnect. Various models have been developed for this purpose, for example [2] for the case of an Ethernet.

6 Conclusions

We have presented an queueing approach for modelling the performance of OPS. The methodology involves standard queueing nodes in a state-dependent fixed point model which is solved by iteration. However, many parameters need to be estimated which are software dependent, arising from the way the database is partitioned amongst instances and how locks are allocated. A two-dimensional representation is required which will show the nature of the sharing of each datablock amongst all instances; e.g. datablock 3 is read by instances 2, 4 and 7 but updated by instances 7 and 10. The required information could possibly come out of views like V$BH; this needs further investigation and experimentation which are in progress. One also has to keep in mind the volume of data changed or accessed by the instances. If, for example, updates are issued rarely then the impact of overlapping data is going to be small—in the limit we would have a model for data warehousing

We described a closed model with fixed population of transactions. This is a reasonable assumption in constrained systems and is analogous to the approach used for modelling multi-access systems with a given level of multiprogramming. An open model is straightforward to implement in a similar fashion where the response time of the lock server is taken to be proportional to the response time of the whole, open system. A slight complication is the fact that the total number of transactions is no longer constant. Hence the probabilities of encountering lock conflicts have to be recalculated at every step of the iteration.

This paper has described work in progress which has so far detailed the design of a model of OPS. The next steps will be to parameterise the model by detailed measurement using the V$ tables, to implement and validate. Validation will be partly against simulation but really needs a controlled distributed environment for testing with OPS itself running on real hardware. Various partitioning strategies, OPS parameterisations and hardware configurations can then be compared quantitatively.

References

[1] A. Thomasian and I.K. Ryu, *Analysis of database performance with dynamic locking*, Journal of Association for Computing Machinery , vol.37 no.3 3 July 1999, p.491

[2] A.J. Field, P.G. Harrison and J. Parry, *Response times in client-server systems*, in Proc. Tools and Techniques for Performance Evaluation, Palma de Mallorca, 1998.

[3] Günther Stürner, *Oracle7 A user's and Developer's Guide*, Thomson 1995, 1-850-32118-3

[4] P.G. Harrison and N.M. Patel, *Performance modelling of communication systems and computer architectures*, Addison-Wesley, 1993.

[5] *Oracle Parallel Server 7 and 8 Concepts*, Oracle documentation is available at `http://technet.oracle.com/`

[6] Erhard Rahm, *Mehrrechner-Datenkanksysteme*, Addison-Wesley 1994, 3-89319-702-8