Layered Analytic Performance Modelling of a Distributed Database System

Fahim Sheikh, Murray Woodside
Carleton University, Ottawa, CANADA,
email: {sheikhf, cmw}@sce.carleton.ca

Abstract

Very few analytic models have been reported for distributed database systems, perhaps because of complex relationships of the different kinds of resources in them. Layered Queueing Models seem to be a natural framework for these systems, capable of modelling all the different features which are important for performance (eg. devices, communications, multi-threaded processes, locking). To demonstrate the suitability of the layered framework a previous queueing study of the CARAT distributed testbed has been recast as a layered model. Whereas the queueing model bears no obvious resemblance to the database system, the layered model directly reflects its architecture. The layered model predictions have about the same accuracy as the queueing model.

KEYWORDS: performance, analytic modelling, layered queueing modelling, distributed databases

1.0 Introduction

Layered modelling is a new adaptation of queueing models for systems with software servers and software resources. It represents software resources in a natural way so that special approximations do not have to be developed for every system; they are part of the framework. It provides a transparent representation of the software architecture, which makes models easy to develop and understand. We have used layered models mostly to analyze hierarchically distributed systems such as client-server systems, which may have been software bottlenecks [1], and we are currently working on applying layered modelling to network management type applications [2], and for telecommunications software.

Client-server systems frequently have database servers, and one way to make these scale up without bottlenecks is to distribute the data among two or more servers. The present paper considers a small experimental database with two sites, called CARAT, studied by Jenq, Kohler, and Towsley [3], and constructs a layered model of it. We wish to show how the layered modelling framework can be used to model distributed database systems and how a layered model can be developed from the existing data. It shall become apparent that the structure of the layered model closely resembles the authors system architecture.

Up to now, layered models have not been used for systems with peer-to-peer communication, which occurs in the present distributed database system. In a certain sense, peer-to-peer communication violates the basic assumption of layering, that there is an ordered acyclic directed graph of service requests between entities. However, a practical interpretation of peer-to-peer interactions described in this paper permits them to be modelled within the layered paradigm. Similarly, layered models have not previously included locking delays; this paper will show how a representation using a surrogate delay, similar to the approach in other models, including [3], can be used within the layered model.

An advantage of applying layered modelling is the availability of tools that take the layered representation and immediately simulate or give analytic approximations. Rolia and Sevcik described the “Method of Layers” [4] and Woodside et al described a “Rendezvous Nets” solver [5]; the tool that will be used is a hybrid called LQNS [7].

2.0 Layered Queueing Modelling and Database Systems

In this section the fundamentals of the layered modelling approach will be discussed followed by a brief description of the CARAT Distributed Database System. CARAT suited our purposes because it included a wide range of database mechanisms and because the published study included a full description of the data and the model.

2.1 Layered Queueing Modelling

In a layered queueing model there are software server tasks, representing operating system processes, as well as the usual servers which model hardware devices such as processors or disks. A software server makes use of other servers, including its host device (the processor it runs on). When a user task makes a request to a server, which in turn makes requests to a lower level server, we have layers of service. The user tasks go through a cycle of thinking, user operations, and requests to the database, which create the workload for the servers. Figure 1 shows a four-layer system with N users and three servers, two of

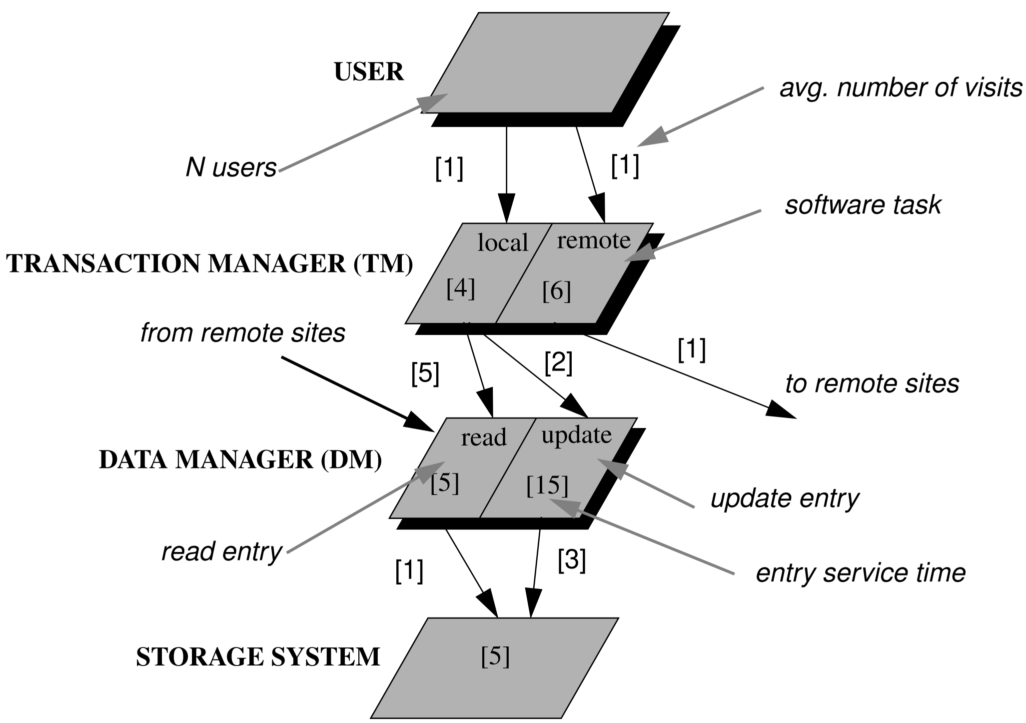

Figure 1 Distributed Database System Architecture, shown as a layered model

which makes requests to one another. Tasks are shown as parallelograms in a notation which follows Buhr [6].

Requests are made to service "entries", which are ports or addresses of particular services offered by a process. Each entry has a characteristic service demand and pattern of requests to other servers. In Figure 1, for instance, the *read* entry on the DM server makes a single disk request to read data, while the *update* entry makes three disk requests because it has to read the data, write it back, and finally write a log entry for recovery. The mean number of service requests to another entry are shown on the arc between the entries. The execution (CPU) demand, which is shown in brackets within each entry, is similarly greater for *update* than for *read*.

A blocking request to a server is a basic interaction in layered systems. Frequently, it is a remote procedure call. If the requesting task has only one thread of execution and it blocks, it has to wait until it gets back the reply. If it has additional threads, they can continue unless they also block. The *service time* of a thread includes its blocked time. One aspect of layered systems that is sometimes hard to understand is that service times are not constant. As traffic level rises, service times tend to increase and saturation may be explosive. Non-blocking interactions can also be represented, but they are not a feature of the present study.

The queueing network solvers for layered models, such as SRVN [6], MOL [4] and LQNS [7] use submodels created for each layer of servers. The users are in the top layer and the devices are in the bottom layer. Each layer submodel includes surrogate delays (see e.g. [8]) to represent the effects of layers above or below it. The layer submodels are solved iteratively by approximate mean value analysis. The tools also include simulators.

2.2 Distributed Databases

A common view of a distributed database system architecture is a set of four layers similar to those shown in Figure 1, with tasks in the layers being replicated over different sites. The User task provides the query interface and communicates through a local Transaction Manager (TM), which breaks the transaction into data queries and maintains transaction properties. The data queries go to Data Managers (DM) at several sites as necessary, and they manage the consistency of local data. The storage level functions include file service and lock management. The CARAT system also has this structure.

Performance constraints arise from contention for three kinds of resources used by the system, namely devices (limited processor and disk speeds), software resources (limited task threads), and locks in the database. The relatively few analytic performance models of distributed databases all concentrate on analyzing contention for devices and in particular for locks, for example papers by Tay [9], and Ryu and Thomasian [10]. It seems that only the layered model can add thread limitations as a performance factor.

To obtain the model parameters for a task such as the TM, one can study the task by itself. For example, the visit rate parameters on the arcs going from TM to the local and remote data managers are the mean number of requests made to each data manager, each time a remote request is made to the TM. The CPU demand is the mean total demand in handling the request.

The CARAT queueing model was analyzed by a similar approach. The *phase of process-*

ing is the state of a Markov Chain, with movements back and forth between states for requests and replies based on arc probabilities. These probabilities are used to determine the number of visits from one phase to another. In the layered model a task was provided for each major phase. Using the number of visits to the phases and the average number of visits to LQN tasks, service times of the LQN tasks can be determined by a method illustrated in Figure 2. In Figure 2(a) we see a simple Markov Chain, with four states (phases)

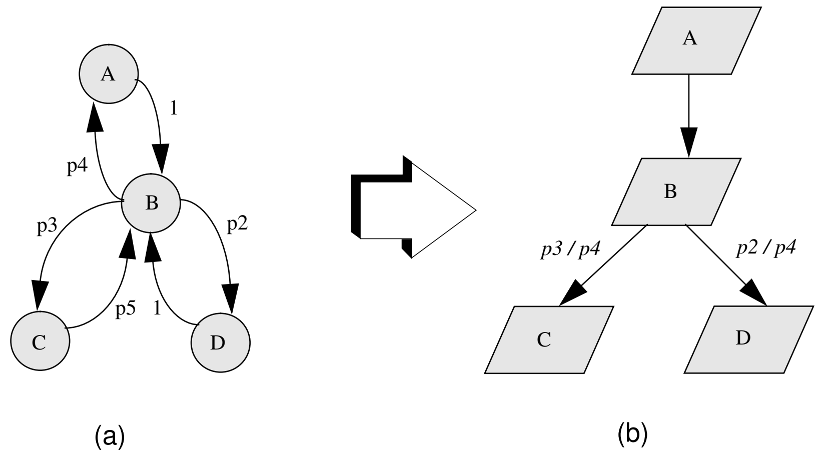

Figure 2 (a) Markov Chain (b) Layered Queueing Model derived from (a)

and with transition probabilities labelled on the arcs. In Figure 2(b) we see the corresponding LQN model.

The arc from task B to C, (taken as an example) represents both the requests from phase B to phase C on the left and the replies from phase C to B. The average number of requests to C, per request to B, is found by elementary analysis to be *p3 / p4* (the probability of the request, divided by the probability of ending the service at B by returning to A).

2.3 The CARAT Distributed Database

CARAT was a two site distributed database system created for empirical performance studies in databases and distributed systems, by Jenq, Kohler and Towsley in [3]. CARAT contains the major functional components of a distributed transaction processing system. Their view of the system architecture is given in Figure 3. Clearly it is very similar to the general layered model in Figure 1.

The phases in processing, represented as states in a Markov Chain, are shown in simplified form in Figure 4(a). The simplifications are to remove the deadlock/abort path from LR to U (and treat aborted transactions as occurring simply with a probability, as was also done in [3]), and to combine the “commit” path from TM to U, at the end of a transaction, with the arc from TM back to U. This assigns the commit workload to U.

In the Markov Chain, the U state is the initial phase. A client request causes a transition to the Transaction Manager (TM) phase. If it is a local transaction then the Database Manager (DM) handles the request while remote requests enter the Remote Waiting (RW) phase. The RW state corresponds to the waiting of a transactions to be completed at the remote site.

The parameters of the Markov Chain are functions of the following system parameters:

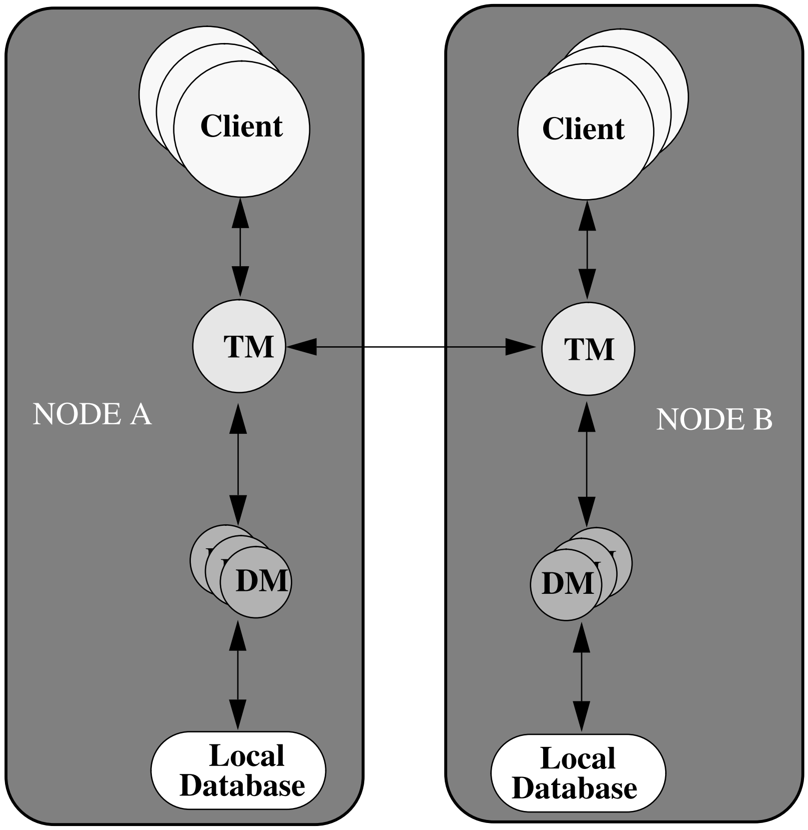

Figure 3 CARAT Process and Message Structure

n = database requests per transaction *($n = l + r$)*

l = local requests (for a local transaction, $n = l$)

r = remote requests (for a distributed transaction they made $r = l$)

q = granules (and locks) per request

P_b = probability that a lock request fails

The examples studied here used the values n = 4 to 20, and q = 4.

The CARAT queueing model had the structure shown in Figure 5. The servers at the bottom of the figure are physical devices, the others are surrogate delays for various functions (TM for TM server serialization; RW for remote requests, CW for commit waiting, and LW for lock waiting). In [3] the TM delay was set to zero and the computation of the CW delay was not given, so we used a value of zero in the layered model.

3.0 Developing the CARAT Layered Model

To develop the layered model in Figure 4(b), a task was created for each phase in Figure 4(a), with an additional "task" DMIO_DISK for disk I/O. Request/reply arcs were created between U and TM, and between TM and DM, corresponding to pairs of transitions for requests and replies between phases. However the task DM was structured to directly invoke the LW, LR, DMIO and DMIO_DISK tasks (in the order shown by the phase tran-

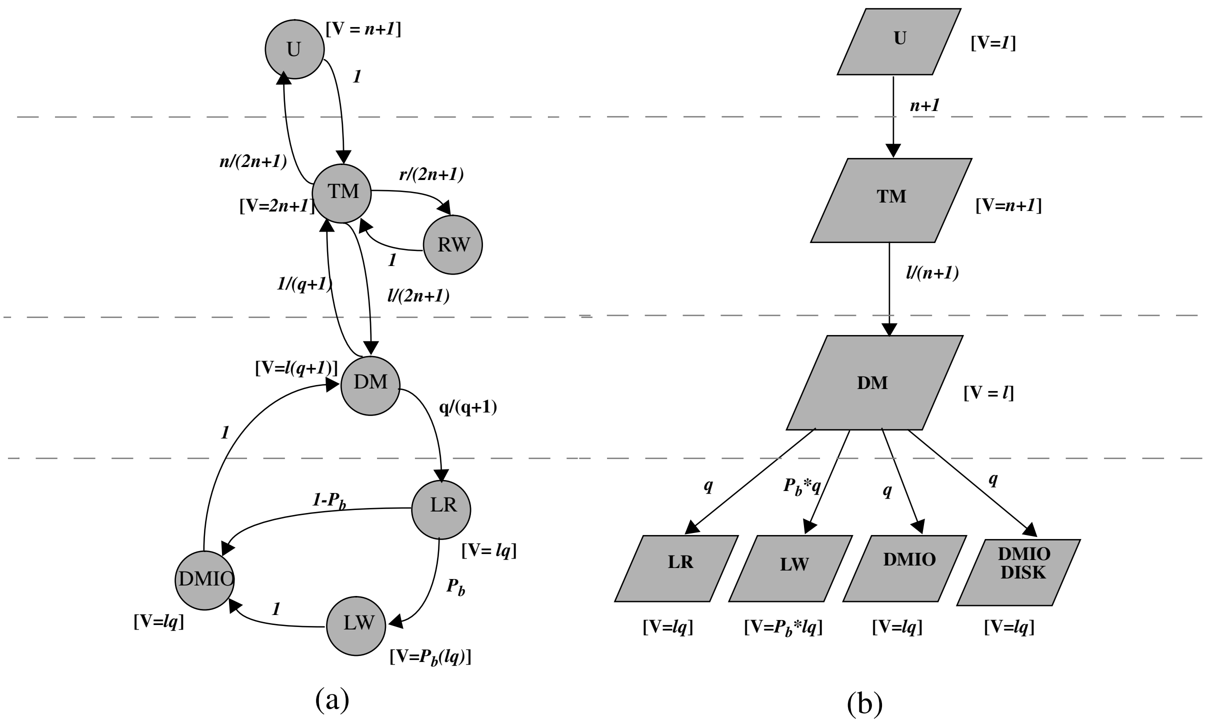

Figure 4 Reduced CARAT Markov Chain and Layered Model for a LRO Transaction

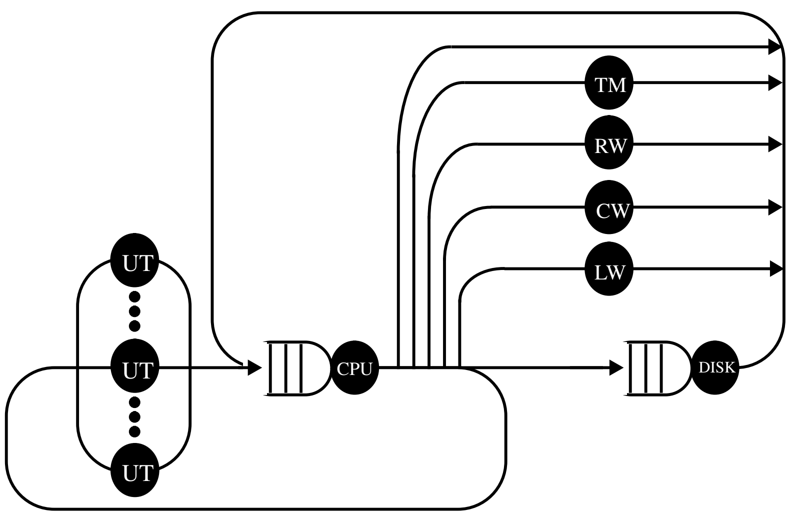

Figure 5 CARAT Queueing Model from [3]

sitions). Figure 4(b) shows one User task, but there were actually four of these in all, for the four transaction classes: Local Read Only (LRO), Local Update (LU), Distributed Read Only (DRO) and Distributed Update (DU). The local transaction requests remain on the originating node, while distributed transactions are composed of both local and remote requests.

An LRO transaction proceeds as follows. In a single cycle, the *User* task executes one transaction. The LRO User task executes a read-only transaction, and the LU User task executes an update transaction. In either case it makes $n+1$ requests to the *Transaction Manager* (TM), where n is the size of the transaction in records, and TM handles each record request in turn. The last request is for committing the transaction. The TM in turn makes requests to the *Database Manager* (DM) for the data, and for each one, the DM must access q number of records. The DM in turn makes requests to obtain locks from the *Lock Request Manager*, (LR). Then because the unit of locking is a disk page, it makes q requests also to the storage system (DMIO and DMIO DISK). An update makes $3q$ requests to DMIO and DMIO DISK, to read, write and log the data.

The execution demands and request parameters of the tasks are given in Tables 1 and 2 below. The arc labels giving the service requests per task invocation were calculated as shown above in Figure 2, for tasks U and TM. The requests from task DM were calculated by making the number of visits to LR, LW, and DMIO per transaction the same, in both models.

The execution demands were calculated from visit counts to phases (shown in square brackets in Figure 4(a), as in [$V_{LR} = lq$]) and execution demands per visit. The visits to each task are also given similarly in Figure 4(b); they are not the same as for phases because for U, TM and DM the phase visit counts include the replies coming up from below as additional visits, whereas the task visit counts do not. In [3] the execution demands are given for one visit to each phase and sometimes are different for each class of transaction. The values are shown in Table 3 as values of $R_{P,C}$ where P is the Phase, and C is the Class. To get the task entry service times we multiply them by the ratio of visits per transaction in the two models, to get the same total work per transaction:

$$S_{T(P),\,E(C)} = R_{P,\,C}\left(\frac{V_{T(P),\,E(C)}}{V_{P,\,C}}\right) \tag{1}$$

where T is the Task name in the LQN model corresponding to phase P and E(C) is the entry corresponding to class C.

Figure 4(b) gives only an outline of the architecture of the complete system, and it does not show the handling of remote requests. The peer-to-peer communications between the

TM's, indicated in Figure 2, was handled by the transformation shown in Figure 6.   It

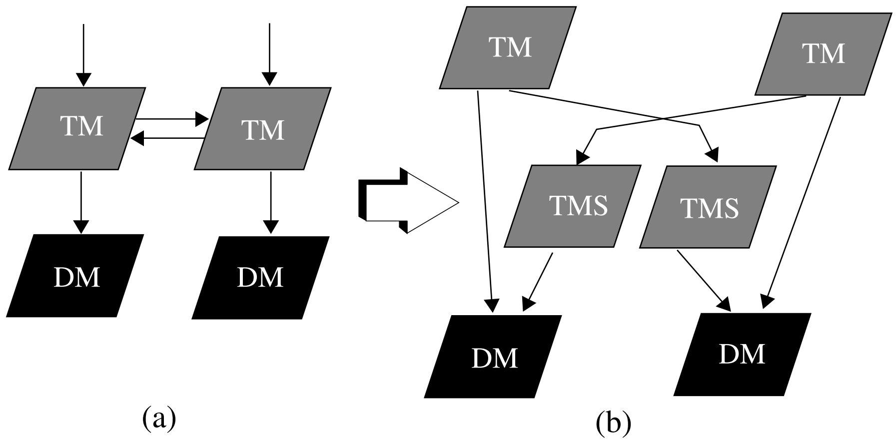

Figure 6 Modelling of peer-to-peer interactions

introduces an asymmetry in the handling of local and remote requests. On the right side of Figure 6 we see local requests going directly from TM to DM at the same site, but remote requests go to a special separate version TMS of the other TM, restricted to handling requests arriving from other sites. TMS handles it the same way as TM handles a local request, and does not generate further requests to any other site. In this way infinite loops between sites ("ping-ponging") are excluded from the model. It is our belief that most superficially symmetrical "peer-to-peer" systems are in fact designed this way, so the transformation is not really restrictive.

When TM is transformed this way, and the four different User tasks are defined for the four types of transactions, we get the complete model shown in Figure 7. Each task has a dark shaded area with its name and a light shaded portion for each entry. The entries handle requests according to their classes. TM has four entries, one for each transaction type, but TMS, DM and DMIO only need two (for read and for update transactions).

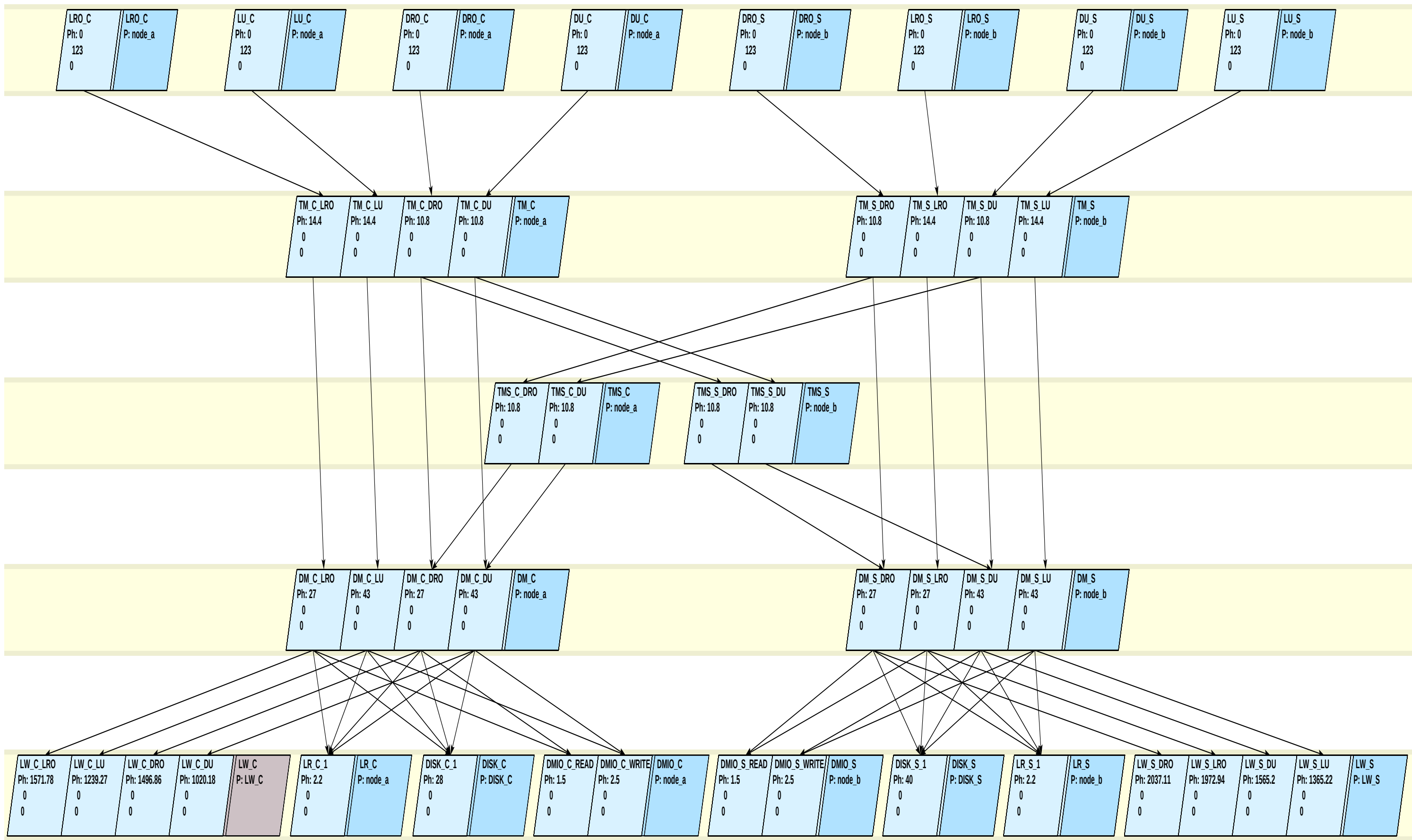

Figure 7 Complete Layered Queueing Model Task Structure

Table 1: Task-to-task Request Counts

FROM Task	FROM Task Entry	TO Task	TO Task Entry	Number of Requests
DRO		TM	DRO	$n+1$
DU		TM	DU	$n+1$
LRO		TM	LRO	$n+1$
LU		TM	LU	$n+1$
TM	DRO	DM	DRO	$\frac{n(1-p)}{n+1}$
	DU		DU	$\frac{n(1-p)}{n+1}$
	LRO		LRO	$n+1$
	LU		LU	$n+1$
TM	DRO	TMS	DRO	1
	DU		DU	1
TMS	DRO	DM	DRO	$\frac{n(1-p)}{n+1}$
	DU		DU	$\frac{n(1-p)}{n+1}$
DM	DRO	LR		q
	DU			q
	LRO			q
	LU			q
DM	DRO	LW	DRO	P_q
	DU		DU	P_q
	LRO		LRO	P_q
	LU		LU	P_q
DM	DRO	DMIO	READ	q
	DU		UPDATE	q
	LRO		READ	q
	LU		UPDATE	q
DM	DRO	DMIO DISK		q
	DU			$3q$
	LRO			q
	LU			$3q$

The service times of the entries of the various processes are summarized below:

Table 2: Service Times in Layered Model (obtained from Table 3 and Eq. (*1*))

Task	Task Entry	Execution Demand
DRO		$7.8 \times (n+1) + R_{COMMIT} = 7.8(n+1) + 84$
DU		$7.8 \times (n+1) + R_{COMMIT} = 7.8(n+1) + 84$
LRO		$7.8 \times (n+1) + R_{COMMIT} = 7.8(n+1) + 84$
DU		$7.8 \times (n+1) + R_{COMMIT} = 7.8(n+1) + 84$
TM or TMS	DRO	$\left(\frac{2n+1}{n+1}\right)\frac{R_{TM,DRO}}{2} = 6\left(\frac{2n+1}{n+1}\right)$
	DU	$\left(\frac{2n+1}{n+1}\right)\frac{R_{TM,DRO}}{2} = 6\left(\frac{2n+1}{n+1}\right)$
TM	LRO	$\left(\frac{2n+1}{n+1}\right)R_{TM,DRO} = 12\left(\frac{2n+1}{n+1}\right)$
	LU	$\left(\frac{2n+1}{n+1}\right)R_{TM,DRO} = 12\left(\frac{2n+1}{n+1}\right)$
DM	DRO	$l(q+1)R_{DM,DRO} = 5.4l(q+1)$
	DU	$l(q+1)R_{DM,DU} = 8.6(l(q+1))$
	LRO	$l(q+1)R_{DM,LRO} = 5.4(l(q+1))$
	LU	$l(q+1)R_{DM,LU} = 8.6(l(q+1))$
LR		$lqR_{LR,\,C} = 2.2lq$
LW	DRO	used lock waiting calculations found in [3], Eq. 12-20, to determine lock waiting times. Each transaction class had a distinct lock waiting time.
	DU	
	LRO	
	LU	
DMIO	READ	$R_{DMIO,READ}$
	UPDATE	$R_{DMIO,WRITE}$
DMIO DISK		$R_{DMIO,DISK}$

Table 3: Queueing Model Execution Demands (msec) obtained from [3][a]

NODE	Class C	$R_{U,C}$	$R_{DM,C}$	$R_{DM,C}$	$R_{LR,C}$	$R_{DMIO,C}$	$R_{DMIO\_DISK,C}$
A	LRO	7.8	8.0	5.4	2.2	1.5	28.0
A	LU	7.8	8.0	8.6	2.2	2.5	84.0
A	DRO	7.8	12.0	5.4	2.2	1.5	28.0
A	DU	7.8	12.0	8.6	2.2	2.5	84.0
B	LU	7.8	8.0	5.4	2.2	1.5	40.0
B	LU	7.8	8.0	8.6	2.2	2.5	120.0
B	DRO	7.8	12.0	5.4	2.2	1.5	40.0
B	DU	7.8	12.0	8.6	2.2	2.5	120.0

a. There is also a demand $R_{COMMIT} = 84$ msec, which is assigned to the User tasks ([11], p. 151).

4.0 Model Solution and Validation

In this section, we discuss the solution results for the CARAT LQN model. Graphs of results for transaction throughput, CPU utilization and disk page throughput are presented. Each graph contains measurement results from the CARAT testbed system, the CARAT Queueing Network Model, and the CARAT LQN model.

As the transaction size increases the probability of a deadlock occurring and in turn a transaction being aborted also increases. In the system studied, it was found that as the transaction size increased beyond *n*=8, there was an increase in the number of aborted transactions. In the CARAT LQN model transaction throughput was adjusted using abort probabilities obtained from [11]. These can be calculated as described in [3]. In the system studied, the only difference between the two sites was that one site had a slower disk than the other.

In all models we see a decrease in transaction throughput as the transaction size increases. Because each transaction is generating more requests, there is an increase in the lock waiting times and an increase in the number of deadlocks resulting transactions being aborted.

Transaction throughput for NODE A and NODE B is presented in Table 4 and Table 5, respectively. Percentage difference calculations w.r.t. to measurements for the Queueing Model and the LQN Model have also been included in the tables.

Table 4: Throughput results for NODE A (Transactions/sec.)

Transaction Size	Testbed Measurement	CARAT Queueing Model		CARAT LQN Model	
4	14.56	1648	+13.1%	18.24	+25.3%
8	16.32	16.96	+3.9%	17.03	+4.4%
12	14.40	14.88	+3.3%	14.98	+4.0%
16	12.16	12.16	0.0%	13.13	+8.0%
20	10.40	8.80	-15.4%	11.78	+13.3%

Table 5: Throughput results for NODE B (Transactions/sec.)

Transaction Size	Testbed Measurement	CARAT Queueing Model		CARAT LQN Model	
	Throughput	Throughout	% diff	Throughput	%diff
4	11.04	12.32	+11.6%	14.11	+27.8%
8	12.80	12.80	0.0%	13.51	+5.5%
12	11.04	12.00	+8.7%	12.07	+9.3%
16	10.88	10.24	-5.9%	10.00	-8.0%
20	8.80	8.00	-9.1%	9.04	+2.7%

In Figure 8 and from the percentage difference calculations in Tables 4 and 5, the errors between predictions and measurement are considerable for both models, with no particular pattern. Some cases have larger errors for the layered model, and some cases have the reverse. All together, the accuracy of the layered approach is comparable to (if a little worse than) the original queueing model. This may be explained by some model details in [3] that we could not completely understand from the paper, and therefore did not include (notably the commit wait phase CW, mentioned above). Other measures besides throughput were better.

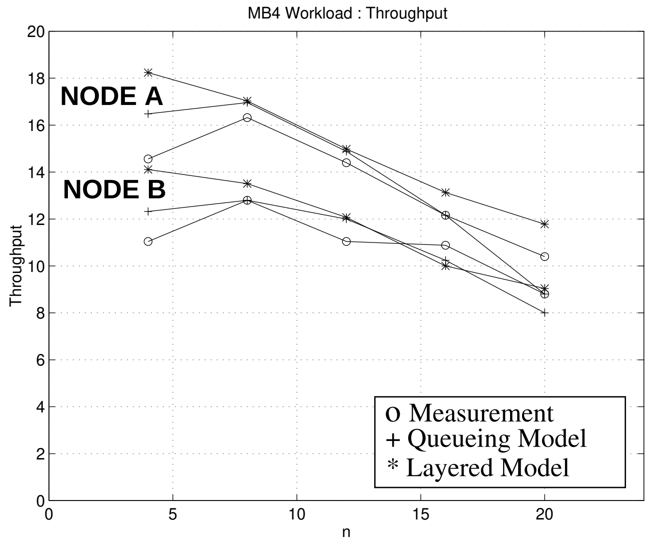

Figure 8 Throughput with adjustment for aborts

The CARAT LQN model was also examined with respect to CPU utilization and disk activity. Figures 9 and 10 compare CPU utilization and disk throughput. The CARAT LQN model better approximates disk throughput and cpu utilization when compared to the queueing network model

Table 6: Disk Activity for NODE A

Transaction Size	Testbed Measurement	CARAT Queueing Model		CARAT LQN Model	
	pages/sec	pages/sec	% diff	pages/sec	% diff
4	27.5	34.0	+23.6%	28.7	+4.4%
8	29.0	32.0	+10.3%	27.9	-3.4%
12	28.0	26.5	-5.3%	27.2	-2.8%
16	26.5	26.5	0%	26.3	-0.8%
20	23.0	27.0	+17.4%	26.3	+14.3%

Table 7: Disk Activity for NODE B

Transaction Size	Testbed Measurement	CARAT Queueing Model		CARAT LQN Model	
	pages/sec	pages/sec	% diff	pages/sec	% diff
4	21.0	24.8	+18.1%	22.0	+4.8%
8	22.5	24.0	+6.7%	21.7	-3.6%
12	22.5	22.5	0%	21.2	-5.8%
16	22.0	21.8	-0.1%	20.7	-5.9%
20	21.8	21.8	0%	20.4	-6.4%

Table 8: NODE A CPU Utilization

Transaction Size	Testbed Measurement	CARAT Queueing Model		CARAT LQN Model	
	Utilization	Utilization	% diff	Utilization	% diff
4	0.43	0.51	+18.6%	0.46	+2.9%
8	0.40	0.44	+10.0%	0.38	-3.2%
12	0.34	0.36	+5.9%	0.35	+3.7%
16	0.30	0.30	0.0%	0.32	0.0%
20	0.25	0.26	+4.0%	0.31	+4.2%

Table 9: NODE B CPU Utilization

Transaction Size	Testbed Measurement	CARAT Queueing Model		CARAT LQN Model	
	Utilization	Utilization	% diff	Utilization	% diff
4	0.35	0.41	+17.1%	0.36	+2.9%
8	0.31	0.35	+11.4%	0.30	-3.2%
12	0.27	0.30	+11.1%	0.28	+3.7%
16	0.26	0.25	-3.8%	0.26	0.0%
20	0.24	0.23	-4.2%	0.25	+4.2%

In Figure 10, CPU utilization decreases as the transaction size is increased. With a larger transaction size we see an increase in lock waiting times. Because of this, different transaction classes are competing less for the cpu.

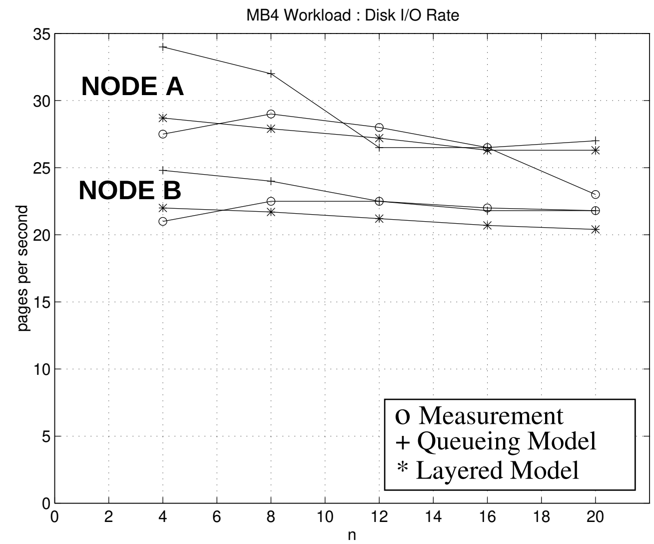

Figure 9 Disk Throughput

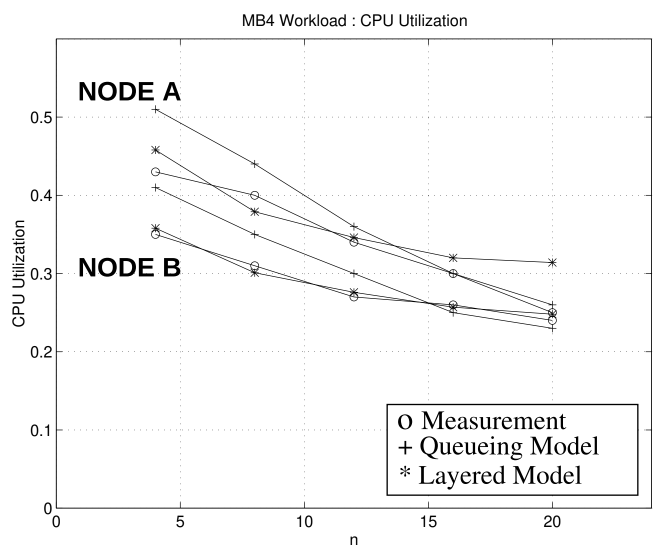

Figure 10 CPU Utilization

As mentioned earlier, the error in the cpu utilization at larger transaction size is due to the fact that LQN model includes the cpu demand for all of a transaction even though the transaction might have been aborted.

In general, the LQN model disk throughput and cpu utilization results are better than the Queueing Network Model when compared to the CARAT testbed measurements at lower transaction sizes.

5.0 Task threads as Resources

It is normal in database software to provide a number of separate threads of control so that several requests can be active at once. The number is usually limited, for two reasons: because each thread requires memory and other operating system resources, and to avoid thrashing at the CPU. Thread level resources are built into layered models, by treating a multi-threaded task as a multiserver in the queueing submodels. If thread levels are unlimited then it is an infinite server, and every request that arrives at the server becomes active immediately; this is the situation implied in the queueing model. If there are M threads and more than M requests, some of the requests suffer queueing delays. Performance restrictions due to inadequate threading levels were studied in [1] under the name of "software bottlenecks".

The results presented in the previous section (and in [3]) were for infinite threading levels. To examine the effect of limitations, the models were solved again with tasks TM, TMS and DM limited to M threads (M = 1, 2, ..., 8). Figure 11 shows the throughputs of the two nodes

.

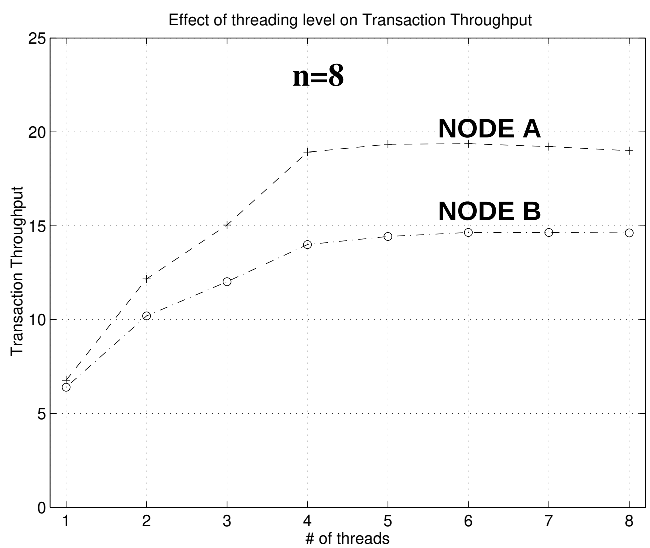

Figure 11 Effect of threading level on transaction throughput

In general, higher threading levels give better performance up to the point where other resources (such as locks, processor, or disk) become saturated. In this case the utilization of the disks on both nodes were 0.91 and 0.84 respectively. From Figure 11, it is apparent that beyond 4 threads there is no gain, which is reasonable given that there are 4 users per node.

6.0 Conclusions

This paper has described how distributed database systems can be modelled using layered queueing networks, and how the layered model differs from the more usual extended queueing network models. It has been argued that the layered paradigm is preferable for ease of understanding and interpretation, and for ease of model-building (because one component can be modelled at a time).

The layered scheme can cover every system attribute included in the queueing model, and some that are not. In the former category is the analysis of the locking delay, which can be taken from existing results and plugged in directly; in the latter category are the representation of the software architecture, and the modelling of task threads as resources with a performance impact.

A new structure was introduced for systems which are designed to provide "peer-to-peer" interactions, to represent them in a layered and thus asymmetrical fashion. The entities are transformed by splitting them, and providing a separate entity to handle a request coming from the peer, so that it does not return to the peer except by way of a reply. This imposes a structure on the interactions. It is conjectured that most if not all practical peer-to-peer-structured software can be modelled by this general approach.

The parameters of the layered model were derived from the previous queueing analysis, showing how the same information is used a little differently. The original CARAT analysis began with a layered structure, which made the task possible. The layered analysis offers the potential advantage that the software tasks can be modelled one at a time (a kind of "divide and conquer"). It was shown that the layered model predictions have about the same accuracy as those from the queueing model.

Finally the built-in capability of the layered model to analyze the effect of threading levels was used to get new results for this model that were not available from the queueing analysis.

The overall conclusion is that layered modelling is a very hospitable environment for distributed database analysis. It appears that much more complex database systems can confidently be modelled this way.

7.0 Acknowledgments

This research was supported by the Industrial Research Chairs program of the Natural Sciences and Engineering Research Council of Canada (NSERC) and the NSERC Graduate Scholarship Program. .

8.0 References

[1] J.E. Neilson, C.M. Woodside, D.C. Petriu and S. Majumdar, "Software Bottlenecking in Client-Server Systems and Rendezvous Networks", IEEE Transactions on Software Engineering, vol. 21, no. 9, pp. 776-782, September, 1995.

[2] F. Sheikh, J.A. Rolia, S. Frolund, P. Garg, A. Shepherd, "Layered Performance Modelling of a Large Scale Distributed Application", in preparation , October 1996.

[3] B.-C. Jenq, W.H. Kohler, D. Towsley, "A Queueing Network Model for Distributed Database Testbed System", IEEE Trans. on Software Engineering, vol. 14, no. 7, pp. 908-921, July, 1988.

[4] J.A. Rolia and K.C. Sevcik, "The Method of Layers", IEEE Trans. on Software Engineering, vol. 21, no. 8, pp. 689-700, August, 1995.

[5] C.M. Woodside, J.E. Neilson, D.C. Petriu and S. Majumdar, "The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-like Distributed Software", IEEE Transactions on Computer, vol. 44, no. 1, pp. 20-34, January, 1995.

[6] R.J.A. Buhr, "Practical Visual Techniques in System Design", Prentice Hall, 1990.

[7] R.G. Franks, A. Hubbard, S. Majumdar, J.E. Neilson, D.C. Petriu, J. Rolia and C.M. Woodside, "A Toolset for Performance Engineering and Software Design of Client-Server Systems", Performance Evaluation, vol. 24, no. 1-2, pp. 117-135, November, 1995.

[8] E.D. Lazowska, J. Zahorjan, G.S. Graham and K.G. Sevcik, "Quantitative System Performance", Prentice-Hall, Inc., Englewood Cliffs, N.J. 07632, 1984.

[9] Y.C. Tay, "Locking Performance in Centralized Databases", Academic Press, Orlando, Fla., 1987.

[10] I. K. Ryu and A. Thomasian, "Analysis of Database Performance with Dynamic Locking", J. of the Association for Computing Machinery, vol. 37, no. 3, July, 1990.

[11] B-C Jenq, "Performance Measurement, Modelling, and Evaluation of Integrated Concurrency Control and Recovery Algorithms in Distributed Database Systems", PhD thesis, University of Massachussetts, 1986.