A QUEUEING NETWORK MODEL FOR A DISTRIBUTED DATABASE TESTBED SYSTEM

Bao-Chyuan Jenq, Walter H. Kohler, and Don Towsley

COINS Technical Report 87-122
November 23, 1987

A Queueing Network Model for a Distributed Database Testbed System[1]

Bao-Chyuan Jenq[2], Walter H. Kohler[3], and Don Towsley[4]

University of Massachusetts
Amherst, MA 01003

ABSTRACT

A queueing network model to analyze the performance of a distributed database testbed system with a transaction workload is developed and validated against empirical measurements. The model includes the effects of the concurrency control protocol (two-phase locking with distributed deadlock detection), the transaction recovery protocol (write-ahead-logging of before-images), and the commit protocol (centralized two-phase commit) used in the testbed system.

The queueing model differs from previous analytical models in three major aspects. First, it is a model for a distributed transaction processing system. Second, it is more general and integrated than previous analytical models. Finally, it reflects a functioning distributed database testbed system and is validated against performance measurements.

Index Terms - queueing network model, performance evaluation, distributed transaction processing, concurrency control, write-ahead-logging, two-phase commit, model validation

[1]This work was supported by the National Science Foundation, grant numbers ECS-8120931, ECS-8340029, and SDB-8418216.

[2]Department of Electrical and Computer Engineering. Now with Microelectronics and Computer Technology Corporation (MCC), Austin, Texas.

[3]Department of Electrical and Computer Engineering.

[4]Department of Computer and Information Science.

1 Introduction

This paper describes an analytical model of a distributed database testbed system with a transaction workload and the validation of the model using performance measurements. The objective of this paper is to show that an analytical model that takes into account the effects of the concurrency control, journaling, and two-phase commit protocols can be developed for a real distributed system executing a variety of transaction workloads. The workloads consisted of both local and distributed, read and write transactions.

Performance modeling studies of some aspects of distributed database systems have been reported previously. For example, Ries was among the first to do a simulation study on the effects of concurrency control on the performance of a distributed data management system [RIES79a]. Garcia-Molina [GARC79] used both simulation and analytical models to study performance of several concurrency control algorithms for fully-replicated distributed databases. Galler [GALL82] analyzed distributed concurrency control algorithms using a simulation model and showed that the performance of basic timestamp ordering is better than that of two-phase locking. Nakanishi and Menasce [NAKA82] used an iterative procedure to analyze a two-phase commit based concurrency control algorithm for a distributed database system. In addition here has been extensive work done on modeling concurrency control algorithms in centralized database systems [RIES77, RIES79b, IRAN79, POTI80, MENA82, THOM82, CHES83, FRAN85, TAY85]. However, the modeling results have frequently been contradictory. In order to explain these differences, Agrawal, Carey, and Livny [AGRA85a] performed a number of concurrency control performance studies using a queueing network simulator. They concluded that many of the differences can be attributed to modeling assumptions that have no clear physical meaning.

There are several important topics that have received little consideration in previous performance modeling studies:

- The resource requirements (CPU, I/O, memory) for the basic components of different concurrency control and recovery algorithms are not well known. Consequently, modeling studies have often neglected the resource requirements when studying or comparing algorithms. Stonebraker [STON83] provided some data on experiments with a prototype distributed INGRES system, but the experiments were not designed to investigate the resource requirements of the concurrency control and recovery mechanisms. Kronenberg et al. recently described the basic performance of the VAX/VMS Distributed Lock Manager in [KRON86].

- Although concurrency control and recovery mechanisms are intimately related, except for the modeling work of Agrawal and DeWitt [AGRA85b], they have been treated as two separate problems. When studying concurrency control, the costs of journaling and rolling back transactions is usually ignored. Agrawal and DeWitt establish a good case for why they must be studied together when investigating their influence on database system performance.

- The impact and validity of the various assumptions made in the models have not been carefully examined. For example, most of the analytical models for locking assumed that only exclusive locks were requested, while the majority of locks are shared locks in real applications. In addition, most of the studies either ignored the effects of deadlocks or assumed a static locking scheme to prevent deadlocks. Dynamic locking and deadlock detection are employed by most locking schemes in commercial database systems.

We have studied the performance of integrated concurrency control and recovery algorithms using both a distributed database testbed system and an analytical model. This paper concentrates on our analytical modeling results and the comparison of those results to our testbed measurements. Our modeling assumptions were made to match the testbed system implementation and synthetic workload, but they can be changed to model other systems. An overview of the distributed testbed system and transaction workload is given in Section 2. Section 3 presents the framework of the queueing network model and its assumptions. The Site Processing Model is described in Section 4. Section 5 presents the derivation of the resource demands for each transaction type in the workload. Section 6 describes a solution procedure for the overall model, illustrates some of the modeling results, and validates the model by comparing its predictions with our testbed performance measurements. Finally, a summary of our conclusions is given in Section 7.

2 CARAT: A Distributed Database Testbed System

A distributed testbed system, called CARAT, has been designed and implemented as a tool for empirical performance studies in database and distributed systems. In this section, we provide an overview of the process and communication structures of CARAT. A more detailed description of the testbed system and the results of our performance experiments can be found in other publications [JENQ86, KOHL86a, KOHL86b].

Following the distributed transaction processing model described in [BERN82], each node in CARAT consists of two levels of server processes. The top level server process, called the TM server, is the Transaction Manager. At the next lower level there is a pool of Data Managers, designated as the DM servers. (The DM servers currently implement a simple CODASYL database system.) The TM and DM servers work cooperatively to service transaction requests issued by user application processes. There is one TM server at each node in CARAT, while the number of the DM servers at each node is fixed and determined at system start-up time. This architecture supports the efficient execution of distributed transactions. (A general discussion of transaction management issues and architectures can be found in [CERI84].)

Figure 1 illustrates the process and message structure of CARAT for two nodes, but the architecture generalizes to any number of nodes. The TM and DM servers work cooperatively to service transaction requests from user application processes. Each user process, labeled TR, submits transaction requests sequentially. In our experiments, the processes at each node were run on a single processor.

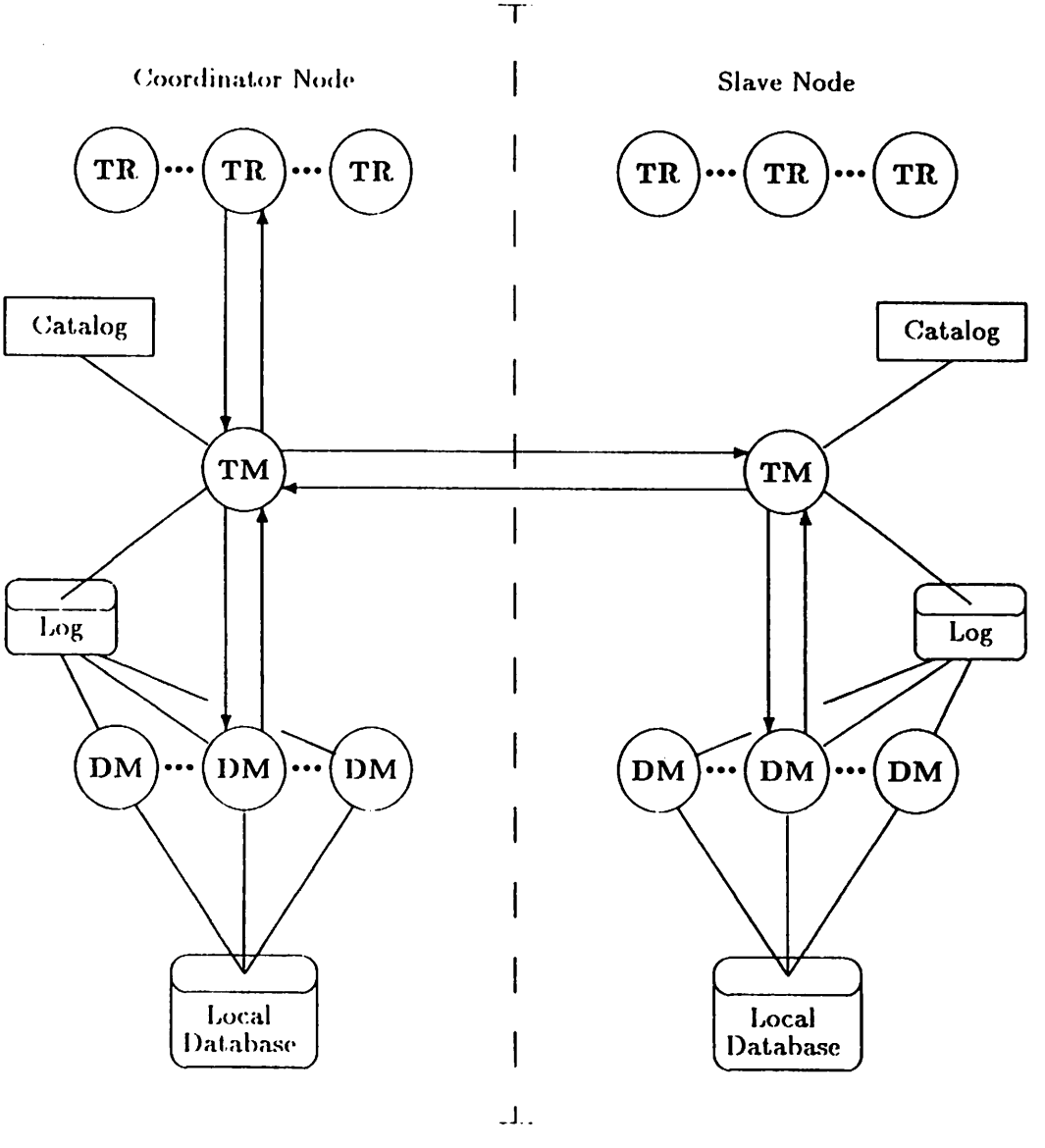

Figure 1: CARAT Process and Message Structure.

One function of the TM servers is to act as the inter-process communication agent for user processes. A user application process, TR, issues transaction requests only to its local TM server process. Moreover, to minimize connect and disconnect overhead associated with communication links, inter-node communication links are established at system start-up time and are maintained only among the TM servers. Each request to a remote DM server is routed through a remote TM server. A local request of a transaction is routed by the local TM server to a local DM server while a remote request is sent to a remote TM server. The DM servers execute the transaction requests and send response messages to their local TM server. For responses to local requests, the TM server then routes the messages to the initiating user process, while responses to remote requests are routed to the user process through the initiating TM server. A DM server is dynamically allocated from the DM server pool by the TM server to a particular transaction for the lifetime of the transaction. Consequently, for a distributed transaction there is a DM server that acts as its agent at each of the participating nodes. In the experiments that we have conducted, only one server at a time can be active for each transaction.

CARAT contains the major functional components of a distributed transaction processing system. In this study, a two-phase locking scheme with distributed deadlock detection was used for concurrency control. Local deadlocks were detected by searching the transaction-wait-for graph, while global deadlocks were detected using a variation of the probe algorithm proposed by Chandy and Misra [CHAN83]. Database access, locking, and logging were done at the physical block level. Before-image journaling was used for transaction recovery and a two-phase commit protocol for global consistency of distributed transactions.

Transactions are composed of sequences of host language statements, including terminal input/output statements, and database DML statements bounded by the transaction delimiters TBEGIN and TEND. In a distributed database system based on message-passing, each of these DML commands may be carried by a message to a DM server which will interpret and then execute the command. This approach requires one request/response message per DML command in addition to the processing overhead associated with the run-time interpretation of the command. To reduce message overhead, several DML commands with parameters can be grouped into a *request* and carried by a single message. This is sometimes called the "function request shipping approach" [CORN86]. To further reduce the run-time interpretation overhead, the requests were pre-compiled into a small object module and stored with the DM server. Requests are invoked at run time by a message that carries the request number and the actual parameters used by the request. Requests are the basic units of execution and distribution among participating nodes.

There are five basic message types issued by a transaction: TBEGIN, DBOPEN, TDO, TABORT, and TEND. A user TR process initiates a transaction by sending a TBEGIN message to its local TM server. This TM becomes the Coordinator TM for the transaction. A TBEGIN message is followed by a DBOPEN message and a sequence of TDO messages, each of which contains a transaction request. A DBOPEN message carries the description of the data objects (e.g., name of the database area or file) upon which the subsequent TDO messages will operate. Upon receiving a DBOPEN message, the TM server determines the location of the data object by consulting its local catalog. The catalog is replicated at each site. If the data object resides locally, it will allocate a local DM server to serve the transaction if one is

not already allocated and then forward the DBOPEN request to the DM server. For remote data objects, the TM forwards the DBOPEN message to the TM server where the data object resides. The remote TM server is called the Slave TM. Transactions initiated at another node are called foreign transactions for the Slave TM server. When a Slave TM receives a request from a foreign transaction, it assigns it a DM server, if it does not have one already, and then forwards the request. After processing the request, a DBOPEN_K acknowledgment is returned to the transaction.[5] One of the parameters included in the DBOPEN_K message is the location of the data object for use in subsequent TDO requests. The use of a catalog by the TM coordinator provides *location transparency* to transactions. This allows transparent file migration and simplifies application programming.

The TDO message carries a request number, a location indicator, and the actual parameters for the request. Each TDO message is routed by the coordinating TM server to either a local DM server, using a DOSTEP message, or a remote TM server, using a REMDO message. For TDO messages which expect data to be returned to the requesting transaction, an acknowledgment message, TDO_K, will carry the response data. Our current implementation assumes that the entire response set can fit into the a single message.

The user process issues a TEND message to commit a transaction. To coordinate execution of distributed transactions, the cooperating TM and DM servers execute a commit protocol. The conventional centralized two-phase commit protocol [GRAY79] has been implemented in CARAT. For each transaction, the Coordinator TM is the the controller. The coordinator communicates with the participating slave TM's to reach unanimous agreement on the outcome (commit, or abort and rollback) of the transaction. During execution of the commit protocol, log records with recovery information are written to the stable storage so that the effects of the transactions can be correctly recovered from system failures in which the volatile memory is lost.

In order to perform performance measurement and modeling studies, we developed a parameterized set of synthetic transactions. Transactions are classified into four basic types: local read-only transactions (LRO), local update transactions (LU), distributed read-only transactions (DRO), and distributed update transactions (DU). A transaction is further parameterized by the number of database requests, i.e., the number of TDO messages, that it issues and the number of database records accessed per request. Each request accesses a fixed number of database records. The records are chosen randomly from among all the database records located at the site. To simplify our tests, we also assumed that the database was partitioned equally among the sites. For the experiments reported here, each request randomly accesses four database records and the number of requests per transaction (called the transaction size and denoted as n) is the same for each transaction in the workload.

Based on these synthetic transactions, several standard workloads were specified. The workloads were chosen to investigate the sensitivity of the results to transaction characteristics. The four workloads used in the two-node tests are:

- LU8: a local-only workload with eight users and a mix of read and write transactions.

[5] The "_K" on a message name is used to indicate an acknowledgment message.

Four users run local read-only (LRO) transactions and the other four users run local update (LU) transactions.

- MB4: a distributed workload with four users at each node and a mix of both local and distributed and read and write transactions. There is exactly one user for each transaction of type local read-only (LRO), local update (LU), distributed read-only (DRO), and distributed update (DU).
- MB8: a distributed workload like MB4, but with a total of eight users at each node consisting of two users of each transaction type.
- UB6: a local-intensive, distributed workload with six users at each node. Two users run local read-only (LRO) transactions, two run local update (LU) transactions, the fifth runs a distributed read-only (DRO) transaction, and the sixth a distributed update (DU) transaction.

Numerous experiments were carried out using these workloads. Measurements from these experiments were then used both to parameterize the models and validate the model predictions. The hardware configuration used in the experiments consisted of two VAX 11/780s, referred to as Node A and Node B, each of which had 6 megabytes of main memory. They were connected by a 10 Mbits/second Ethernet. The size of the database file at each node was 3,000 disk blocks. Each disk block contained 512 bytes and stored six database records. The disk block was the unit of transfer between the CPU and disk. The database disk on Node A was a DEC RM05 and on Node B a DEC RP06. Each node had a separate system disk, but space on this disk was not available for our files. Consequently, the recovery log file had to be on the same disk as the database. (This would not be done in practice, because a single disk becomes a performance bottleneck and a single disk failure could destroy the database and the recovery log.)

3 The Queueing Network Model Framework and Assumptions

In the following sections, we present a queueing network model for CARAT using two-phase locking for concurrency control and before-image journaling for recovery. The model extends the centralized model used in [IRAN79] and [THOM82] to a distributed transaction processing system and includes the effects of journaling and rolling back transactions due to detected deadlocks. It also takes into account both shared and exclusive locks. Although the model closely reflects the CARAT testbed architecture and transaction execution, the modeling approach can be adapted to model other architectures. The choices made here should not be considered as a limitation of the general modeling approach.

The queueing network model contains two levels. The high-level model, called the Site Processing Model, represents transaction execution at a single site of the distributed CARAT system. The site model represents a single site as a product form queueing network with multiple routing chains [BASK75]. A distributed CARAT system is represented as a set of interacting Site Processing Models. The interactions of the Site Processing Models for distributed transactions

are accounted for by including delay servers at which distributed transactions wait for messages from other sites. The remote requests associated with distributed transactions are modeled as a separate type of transaction, called a distributed slave.

Since the communication delay for inter-site message passing depends on the characteristics of the underlying network and the loads from each site in the system, we employ a low-level model, called the Communication Network Model, to calculate average communication delay, α, for inter-site messages. For example, in an Ethernet network with high contention, the Ethernet model proposed by Almes and Lazowska [ALME79] can be used.

In developing the model, a number of assumptions have been made to match the CARAT testbed implementation, the test database, and the transaction workloads. These assumptions are summarized below:

- The database is partitioned among the sites with no data replication. Each site contains N_g granules where each granule contains N_b records. Transactions access records randomly and uniformly.
- There is at most one request being executed per transaction at any point in time. That is, except for the execution of the commit protocol, concurrent executions of requests for a single transaction at multiple sites are not possible. Also, only a single message is necessary for a request or response.
- Lock requests are uniformly distributed over the lifetime of a transaction and the granularity of a lock request is one database granule, i.e., a database block. The processing of a lock request requires no disk I/O, since the lock table is maintained in main memory.
- There is a finite population of transactions for each transaction type and each transaction issues a fixed number of requests. Furthermore, each request retrieves or retrieves and then updates a fixed number of database records.
- Each database granule used by a transaction requires a disk I/O operation, i.e, a shared database buffer is not used to reduce database I/O.

4 Site Processing Model

The CARAT Site Processing Model, shown in Figure 2, describes the behavior of the transactions at one site in the CARAT system. There are multiple service centers, represented by circles, in the site model and at any point in time a transaction occupies one service center. The two queueing centers, CPU and DISK represent the corresponding physical shared resources. Multiple DISK queueing centers can be used to represent multiple disks for the database or separate disks for the database and recovery log. Only one DISK service center is shown in Figure 2, since this is the configuration used in our empirical studies. (The system disk can be ignored because no system I/O was required during the test.) The Lock Wait (LW), Remote Wait (RW), Two-Phase Commit Wait (CW), and TM Server Serialization (TM) delay centers

are introduced to model various synchronization delays encountered during transaction execution. The User Think-time (UT) delay centers are introduced to incorporate the effects of user think time between successive transaction executions.

A transaction enters the LW delay center whenever one of its lock requests is queued, i.e., the transaction is blocked due to lock conflicts. Similarly, as a transaction executes, it enters the RW delay center whenever it waits for a response for a remote request issued by the TM server on its behalf. Since a two-phase commit protocol is executed at the end of a distributed transaction, the CW delay time is used to capture the synchronization wait time for coordinator-slave communications. In addition to the synchronization delays discussed above, a transaction may experience a serialization delay due to contention for the TM server, since there are multiple concurrent transactions to be serviced by the single server at each site. The TM delay center is introduced to represent this serialization delay.

4.1 Transaction Phases

For each of the service centers, we need to calculate the service demands required during the execution of a transaction. Since at any point in time a transaction in the distributed CARAT system is executed within the context of three types of processes, i.e., User TR, TM server, or DM server, it is convenient to partition a transaction execution into a sequence of transaction phases. As it is executing, a transaction always resides in one of the following phases:

- Transaction initialization processing (INIT): processing requirements for transaction initialization.
- User application processing (U): the processing time required for a user application to prepare and process a request.
- TM processing (TM): the time required for the TM server to process a request, including the time needed to receive and send a local or remote message.
- DM processing (DM): the processing time consumed by the DM server (including I/O processing) between two lock requests.
- DM disk I/O (DMIO): a burst of disk I/O operations.
- Lock request processing (LR): the processing time of a lock request, including local deadlock detection.
- Lock wait (LW): the blocking time for a blocked lock request.
- Remote request wait (RW): the time a coordinator transaction spends waiting for a response to a remote request or a slave transaction spends waiting for the next request from its coordinator.
- Transaction commit processing (TC): the aggregate processing requirements of the two-phase commit protocol, i.e., PREPARE and COMMIT message processing.

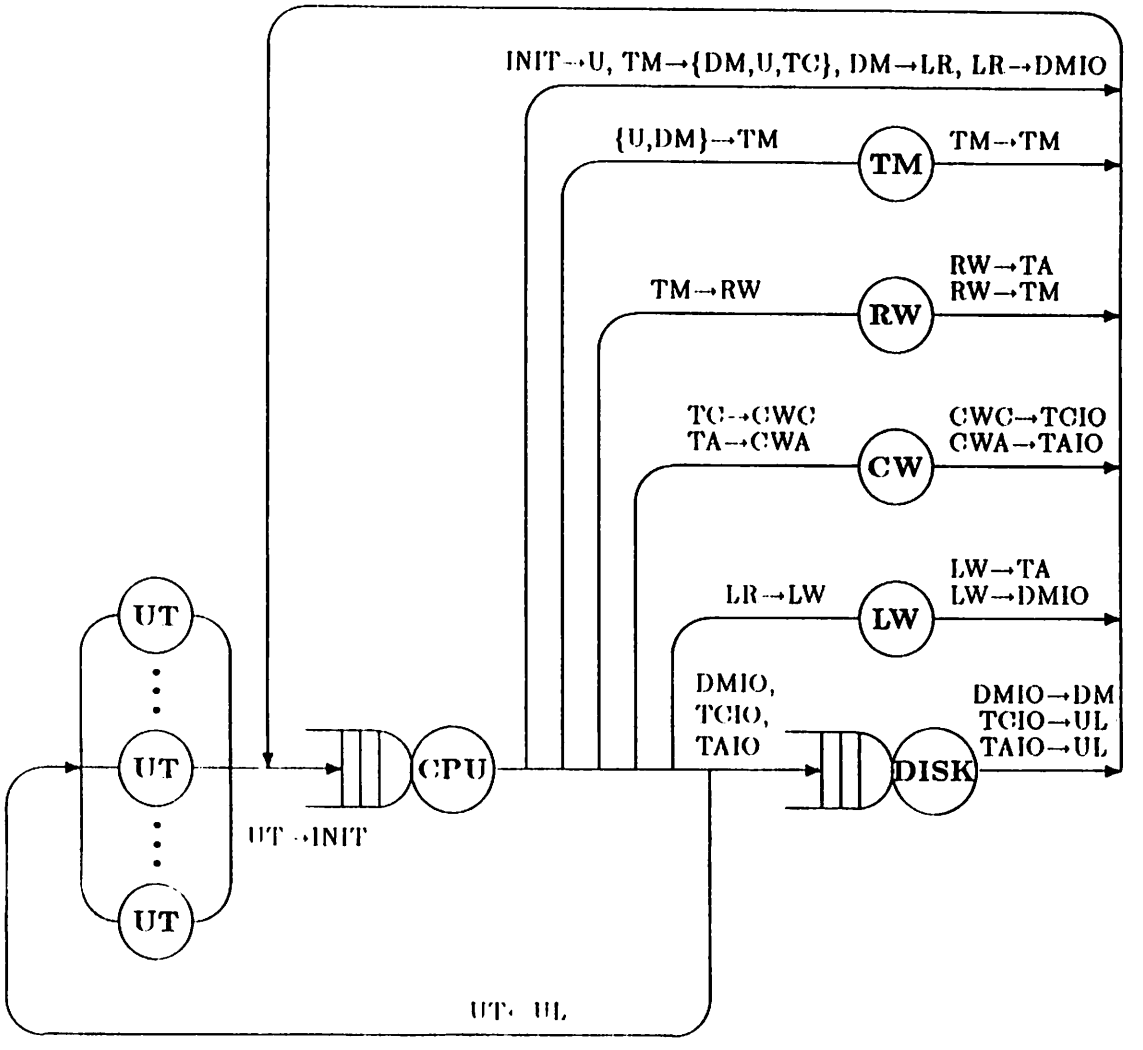

Figure 2: CARAT Site Processing Model.

- Transaction commit disk I/O (TCIO): the disk I/O required to write the commit log records.
- Transaction abort processing (TA): the processing time required to roll back an aborted transaction.
- Transaction abort disk I/O (TAIO): the disk I/O required to roll back a transaction.
- Two-phase commit wait for distributed transactions (CW): This can be further categorized into CWC and CWA phases representing the commit and abort waiting time respectively.
- Unlock processing (UL): the processing time required to release all locks held by a transaction.
- User think wait (UT): the time between the completion of one transaction execution and the initiation of the subsequent execution.

We let P denote the set of phases, i.e., P = { *INIT, U, TM, DM, DMIO, LR, LW, RW, TC, TCIO, TA, TAIO, CW, UL* }.

To illustrate how a transaction changes its phase during execution, consider an example transaction execution. A transaction begins in the initialization (INIT) phase in which TBEGIN/TBEGIN_K and DBOPEN/DBOPEN_K messages are processed. After the INIT phase, a transaction enters the user application (U) phase which represents an execution of the user application process. During the U phase, some amount of the CPU resources are consumed by the user TR process to prepare and send a request to the TM server. The transaction changes to the TM processing phase (TM) when the TM server receives the transaction request and sends either a message to a remote site (REMDO message) or a message to a local DM server (DOSTEP message). In the case of a remote message, the transaction transits to the remote request wait (RW) phase as it is waiting for a response from the remote site. A message sent to a remote site is processed by the TM server at the remote site. This is modeled by a slave transaction at the remote site that makes a transition directly from the user think (UT) phase to the TM processing phase (TM). On the other hand, a DOSTEP message signals the beginning of the DM server processing (DM) phase at the local site. To model the locking activities of the DM server, a transaction in the DM phase may change into the lock request (LR) phase whenever the DM server issues a lock request on its behalf. Depending on whether there is a lock conflict or not, it may enter either the lock wait (LW) phase to wait for the lock or the DM disk I/O (DMIO) phase to access data. Since a transaction in the lock wait phase may become a deadlock victim, it may switch into the transaction abort processing (TA) phase and then transaction abort disk I/O phase to begin a sequence of CPU and I/O operations to roll the transaction back. When a transaction ends normally, it enters the transaction commit processing (TC) phase and begins the execution of the two-phase commit protocol. The two-phase commit wait and abort (CWC and CWA respectively) phases are used to represent the two-phase synchronization time required to commit or abort a distributed transaction. Two-phase commit processing requires CPU time as well as I/O. The CPU time required to

commit or roll back a transaction is represented in the TC (for commit) and TA (for abort) phases respectively, while the I/O time is represented in the transaction commit disk (TCIO) and transaction abort disk (TAIO) phases. A transaction ends in the unlock processing (UL) phase where all its locks are released. After all locks are released, the transaction changes to the user think (UT) phase and waits until the next transaction is initiated.

4.2 Transaction Types

As discussed earlier, transactions in the workload are categorized into four transaction types. It is assumed that there is a finite population of transactions of each type. Local read-only (LRO) and local update (LU) transactions never enter the remote wait (RW) and two-phase commit wait (CW) delay centers, since they do not request remote operations. However, distributed read-only (DRO) and distributed update (DU) transactions perform requests at the coordinator site as well as at the participating slave sites. Therefore, we represent a distributed transaction as a collection of single-site transactions. Local operations of a distributed transaction are executed by a coordinator transaction at the coordinator site, while remote operations are executed by a slave transaction at each slave site. In the Site Processing Model, distributed read-only (DRO) and distributed update (DU) transactions are decomposed into one coordinator transaction, referred to as DROC or DUC respectively, and one or more slave transaction, referred to as DROS or DUS transactions respectively. Thus, we have six transaction types in the model: local read-only (LRO) transactions, local update (LU) transactions, distributed read-only coordinator (DROC) transactions, distributed update coordinator (DUC) transactions, distributed read-only slave (DROS) transactions, and distributed update slave (DUS) transactions. Within the model, we let T denote the set of transaction types, i.e., $T = \{LRO, LU, DROC, DUC, DROS, DUS\}$, and define $N(t,i)$ as the number of type t transactions, $t \in T$, at site i. For example, if site i has four local read-only transactions and is the slave site for two distributed update transactions, then $N(LRO,i) = 4$ and $N(DUS,i) = 2$.

Since distributed read-only slave (DROS) and distributed update slave (DUS) transactions execute remote requests for slave sites that are entered through the network, these transactions have no user think time associated with them. Consequently, the delay incurred at the UT center corresponds to the time that they lie dormant between the completion of one execution and the initiation of the subsequent execution. Furthermore, to a DROS or DUS transaction the remote wait (RW) phase corresponds to waiting for either a subsequent remote request or for a two-phase commit Prepare request from its coordinator. On the other hand, to a distributed read-only coordinator (DROC) or distributed update coordinator (DUC) transaction, the RW phase corresponds to waiting for the completion of a remote request at a slave site.

5 Transaction Service Demands

In this section we concern ourselves with determining the parameters for the queueing network in Figure 2. We first calculate the visit counts, $V_c(t,i)$, $c \in P$, the average number of times that

a type t transaction on site i enters phase c. Following this, we determine the service demands for each transaction type and site.

5.1 Calculation of Visit Counts

Let $p_{c_1,c_2}(t,i)$, called the *phase transition probability*, denote the probability that a transaction of type t on site i enters phase c_2 at completion of phase c_1. We introduce the following notation:

- $l(t)$: number of local requests by a type t transaction;
- $r(t)$: number of remote requests by a type t transaction; (Note that $r(t) = 0$ for $t \in \{LRO, LU\}$);
- $n(t)$: total number of requests by a type t transaction, i.e., $n(t) = l(t) + r(t)$;
- $q(t)$ mean number of disk I/O operations for a request by a type t transaction;
- $C(t)$:total number of transitions out of the TM phase, i.e.,$C(t) = 2n(t) + 1$, ;
- $Pb(t,i)$: probability that a lock request is not granted to a type t transaction at site i;
- $Pd(t,i)$: probability that a blocked type t transaction at site i is chosen as deadlock victim;
- $Pra(t,i)$: probability that a type t transaction at a remote wait center at site i is aborted due to a deadlock detected at the remote site.

When there is no confusion, we will omit the arguments denoting transaction type and site. The phase transition probabilities for local and coordinator transactions are found in Table 1.

As an illustration for the phase transition probabilities, consider the entries for the TM phase. The number of transitions into the TM phase for a type t transaction is $2n(t) + 1$ because, in addition to the TEND message, there are $n(t)$ database requests for a transaction and each request requires two transitions to its local TM server. The first corresponds to the processing of a TDO message from the user application process and the second corresponds to the processing of a DOSTEP_K (or a REMDO_K) message from the local DM server (or from a remote site). A transaction in TM phase may change its phase to U, DM, RW, or TC, depending on whether the transaction is executed next by the user process, a local DM server, a remote server, or it begins the two-phase commit execution. The relative probabilities of entering DM or RW are based on the number of local requests or remote requests by the transaction. Thus, we have $p_{TM,DM} = l(t)/(2n(t) + 1)$ and $p_{TM,RW} = r(t)/(2n(t) + 1)$. A transaction in phase TM changes its phase to U and TC with probabilities of $p_{TM,U} = n(t)/(2n(t) + 1)$ and $p_{TM,TC} = 1/(2n(t) + 1)$ because an execution of the two-phase commit protocol follows a sequence of $n(t)$ requests.

From Table 1, we can calculate the visit counts for an execution of a transaction, which may be committed or aborted. The visit counts satisfy the following equations

$$V_{c_2} = \sum_{c_1 \in P} V_{c_1} p_{c_1,c_2}, \quad c_2 \in P. \tag{1}$$

	UT	INIT	U	TM	DM	LR	DMIO	LW	RW	TC	TA	TCIO	TAIO	CWC	CWA	UL
UT	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
INIT	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
U	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
TM	0	0	n/C	0	l/C	0	0	0	r/C	$1/C$	0	0	0	0	0	0
DM	0	0	0	$1/(q+1)$	0	$q/(q+1)$	0	0	0	0	0	0	0	0	0	0
LR	0	0	0	0	0	0	$1-Ph$	Ph	0	0	0	0	0	0	0	0
DMIO	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
LW	0	0	0	0	0	0	$1-Pd$	0	0	0	Pd	0	0	0	0	0
RW	0	0	0	1 Pra	0	0	0	0	0	0	Pra	0	0	0	0	0
TC	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
TA	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
TCIO	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
TAIO	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
CWC	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
CWA	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
UL	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1: Transaction Phase Transition Probabilities.

Similar expressions for the phase transition probabilities can be obtained for the two slave transaction types.

5.2 Mean Number of Disk I/O Operations per Message Request

We assume that the number of database records accessed by transactions of type t, denoted by $N_r(t)$, $t \in T$, is given as an input parameter. We also assume that multiple database records are grouped together and stored as a database granule and the number of database records per granule, denoted by N_b, is given as an input parameter. The granule is the unit of I/O transfer between the stored database and the processor. If we assume that any database record is equally likely to be accessed by a transaction of type t, then the mean number of granules accessed by a transaction of type t, $g(t)$, can be calculated using the formula described in [YAO77]. (For the workloads used in our performance experiments, $g(t)$ is very close to $N_r(t)$.) Assuming that one disk I/O operation is required for each granule accessed and the mean number of disk I/O operations required for a request is the same for all sites, then the mean number of disk I/O operations required for a type t request, $q(t)$, can be calculated as $q(t)$ $g(t)/n(t)$.

5.3 Calculation of Service Demands

Given the visit counts to the different phases during transaction execution, we can calculate the mean total service demand to each service center between two successive transaction commits, including the service demands of unsuccessful transaction executions. The mean total service

demand is calculated as the product of the mean service demand of a transaction execution and the mean number of transaction submissions for a transaction commit.

Let $N_{lk}(t)$ denote the number of locks requested by a type t transaction. Assuming that one lock is requested for each granule accessed, we have

$$N_{lk}(t) = l(t) \cdot q(t),\ t \in T. \tag{2}$$

We can now calculate $P_a(t,i)$, the probability that a type t transaction at site i is aborted due to a deadlock detected either at that site or at one of the remote sites. We have

$$P_a(t,i) = \begin{cases} 1 - (1 - Pb(t,i)Pd(t,i))^{N_{lk}(t)}, & t \in \{LRO, LU\}, \\ 1 - (1 - Pb(t,i)Pd(t,i))^{N_{lk}(t)} \cdot (1 - Pra(t))^{r(t)}, & t \in \{DROC, DUC\}. \end{cases} \tag{3}$$

The mean number of submissions required for a type t transaction at site i to commit, $N_s(t,i)$, is:

$$N_s(t,i) = \sum_{k=1}^{\infty} k(1 - P_a(t,i))P_a(t,i)^{k-1} = \frac{1}{1 - P_a(t,i)}, \tag{4}$$

where k denotes the number of submissions required between two successive commits.

Let P_{cpu} and P_{disk} denote the sets of all phases during which a transaction uses the CPU and I/O respectively. Let $R_c^{(cpu)}(t,i)$ and $R_c^{(disk)}(t,i)$ denote the CPU and I/O service requirements respectively, when a type t transaction is in phase c. Let $R_{LW}(t,i)$, $R_{RW}(t,i)$, $R_{CW}(t,i)$, and $R_{UT}(t,i)$ denote the delay for each visit to the LW, RW, and CW delay centers respectively. The service demands at the different service centers are

$$D_{cpu}(t,i) = N_s(t,i) \sum_{c \in P_{cpu}} V_c(t,i) R_c^{(cpu)}(t,i),\ \text{CPU demands} \tag{5}$$

$$D_{disk}(t,i) = N_s(t,i) \sum_{c \in P_{disk}} V_c(t,i) R_c^{(disk)}(t,i),\ \text{I/O demands} \tag{6}$$

$$D_{LW}(t,i) = N_s(t,i) V_{LW}(t,i) R_{LW}(t,i),\ \text{LW center demands} \tag{7}$$

$$D_{RW}(t,i) = N_s(t,i) V_{RW}(t,i) R_{RW}(t,i),\ \text{RW center demands} \tag{8}$$

$$D_{CW}(t,i) = N_s(t,i) V_{CW}(t,i) R_{CW}(t,i),\ \text{CW center demands} \tag{9}$$

The total user think time between two successive transaction commits, excluding the think time after a successful transaction execution, is

$$D_{UT}(t,i) = (N_s(t,i) - 1) R_{UT}(t,i). \tag{10}$$

5.4 Lock Wait Delay

In order to calculate the mean lock wait time, we first estimate $Pb(t,i)$, the probability that a lock request of a type t transaction at site i is blocked. The approximation of $Pb(t,i)$ is obtained by considering $L_h(t,i)$, the time-average number of locks held by a type t transaction at site i.. We also derive an approximation of $Pd(t,i)$, the probability that a blocked lock request of the type t transaction at site i is chosen as a deadlock victim. We then show how the mean lock wait time, $R_{LW}(t,i)$, is calculated using the mean execution time of conflicting transactions. Equation (1) can then be used to determine the lock wait delay, $D_{LW}(t,i)$.

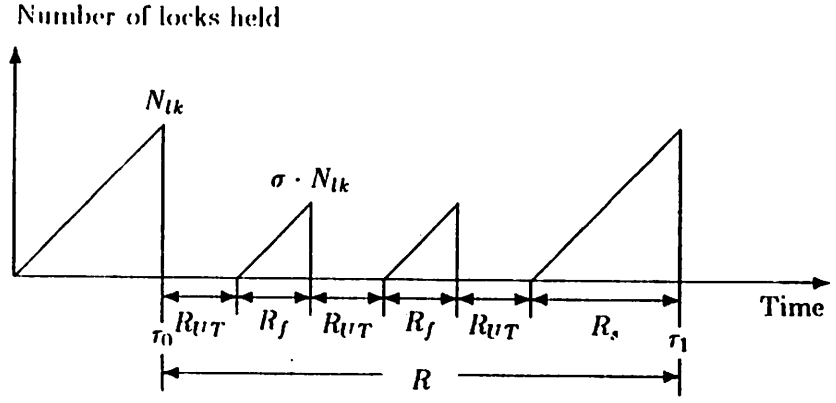

Figure 3: Number of locks held by a transaction vs. time.

5.4.1 Approximation of Average Number of Locks Held

The following derivation of an expression for the time average number of locks held by a transaction, $L_h(t,i)$, is independent of the transaction type and the site. Consequently, we omit the arguments t and i during the remainder of this section. The lock acquisition behavior of a transaction is illustrated in Figure 3. From the time the transaction begins, τ_0, until it completes at time τ_1, it performs several executions resulting in aborts followed by a successful execution. Preceding each of these executions, the transaction incurs a mean user think delay R_{UT}. The parameters R_f and R_s denote the average execution times for an aborted transaction and a successfully completed transaction. R denotes the average transaction response time including aborts and think times.

We are interested in the average number of locks held by a transaction, L_h during its execution. We assume that locks are requested uniformly during the execution of a transaction and that they are released at the end whether it is aborted or completed successfully. That is, locks are accumulated at a fixed rate for a period equal to the transaction execution time.

Let Y denote the number of locks held by a transaction at the end of an aborted execution. The distribution of Y is

$$P[Y = i] = (1 - Pb \cdot Pd)^i Pb \cdot Pd / [1 - (1 - Pb \cdot Pd)^{N_{tk}}], \; i = 0, 1, \cdots, N_{tk} - 1$$

and the expected value of Y is

$$E[Y] = (1 - Pb \cdot Pd)/(Pb \cdot Pd) - (1 - Pb \cdot Pd)^{N_{tk}}/[1 - (1 - Pb \cdot Pd)^{N_{tk}}]. \qquad (11)$$

Define σ as the fraction of the mean number of locks held at the time the transaction is aborted divided by the total number of lock requests. That is, $\sigma = E[Y]/N_{tk}$. Observe that R_f can be expressed as $R_f = \sigma R_s$ as a result of the uniform lock acquisition assumption.

Let $L_h^{(f)}$ and $L_h^{(s)}$ denote the average number of locks held during a single failed execution and successful execution respectively. As a consequence of the uniform lock acquisition assumption, these quantities can be expressed as

$$L_h^{(s)} = \frac{N_{lk}}{2} \times \frac{R_s}{R_{UT} + R_s}, \tag{12}$$

$$L_h^{(f)} = \frac{E[Y]}{2} \times \frac{R_f}{R_{UT} + R_f} - \frac{\sigma^2 N_{lk}}{2} \times \frac{R_s}{R_{UT} + \sigma R_s}. \tag{13}$$

Finally, L_h is expressed in terms of $L_h^{(s)}$ and $L_h^{(f)}$ by removing the conditioning on whether the transaction is successful or not. This results in

$$\begin{aligned} L_h &= |(R_{UT} + R_s)L_h^{(s)} + (R_{UT} + R_f)L_h^{(f)}|/|N_s R_{UT} + (N_s - 1)R_f + R_s| \\ &\approx \frac{N_{lk}}{2} \times \frac{|1 - (1 - \sigma^2)P_a|R_s}{P_a R_f + (1 - P_a)R_s + R_{UT}}. \end{aligned} \tag{14}$$

5.4.2 Approximation of Blocking Probabilities

The calculation of $Pb(t,i)$ proceeds in the following way. A request for a shared lock is blocked if some transaction has an exclusive lock on that granule. A request for an exclusive lock is blocked if the granule is locked by any other transaction. The mean number of granules occupied by type t transactions at site i is given by $N(t,i)L_h(t,i)$ and since a transaction is not blocked by the locks held by itself, we have,

$$Pb(t,i) = \begin{cases} (1/N_g)\left[\sum_{t \in \{LU,\, DRUC,\, DRUS\}} N(t,i)L_h(t,i) - L_h(t,i)\right], & t \in \{LRO,\ DROC,\ DROS\}, \\ (1/N_g)\left|\sum_{s \in T} N(t,i)L_h(t,i) - L_h(t,i)\right|, & t \in \{LU,\ DUC,\ DUS\}. \end{cases} \tag{15}$$

The probability that a type t transaction at site i is blocked due to lock conflicts, $P_{lw}(t,i)$, can be estimated as,

$$P_{lw}(t,i) = 1 - (1 - Pb(t,i))^{N_{lk}(t)}. \tag{16}$$

When $Pb(t,i)$ is small, P_{lw}(t,i) can be approximated as $N_{lk}(t) \cdot Pb(t,i)$.

Let $PB(t,s,i)$ be the conditional probability that a type t transaction is blocked by a type s transaction at site i, given that a lock request of the type t transaction is blocked at site i. That is,

$$PB(t,s,i) = P[t \text{ is blocked by } s \text{ at } i \mid t \text{ is blocked at i}] = \frac{N(s,i)L_h(s,i)}{\sum_{r \in T} N(r,i)L_h(r,i) - L_h(t,i)}. \tag{17}$$

5.4.3 Approximation of Probability of Deadlock

We want to estimate the conditional probability, $Pd(t,i)$, that a transaction of type t is the victim of a deadlock, given that a lock request of the transaction is blocked at i. Since it

has been observed that most deadlock cycles are of length two [GRAY81], we approximate the probability of deadlock by considering only two-cycle deadlocks. Therefore, our estimations for the probabilities of local and global deadlocks are only first-order approximations. Note, however, that by observing the relative frequencies of more-than-two-cycle vs. two-cycle deadlocks in the experiments, we can determine an adjusting factor for each workload. Both local and global deadlocks are considered in our estimate of $Pd(t,i)$. For a two-node system such as our experimental environment, the conditional probability of global deadlocks is approximated by considering only two-cycle global deadlocks involving two distributed transactions. Obviously, this is rather restrictive considering that a global deadlock may involve both local and distributed transactions. However, the probability of global deadlocks is comparatively smaller than that of local deadlocks, the underestimation of $P_d(t,i)$ should not be a significant factor for most workloads. The derivation, shown in [JENQ86], is omitted here.

5.4.4 Approximation of Blocking Time

Let $RLT(t,i)$ denote the average time that a transaction at site i is blocked by a type t transaction. lock. Using renewal theory arguments, we have derived the following expression for $RLT(t,i)$ under the assumption that aborts occur rarely, i.e., $Pd(t,i) \approx 0$ (see [JENQ86]),

$$RLT(t,i) \approx \frac{2N_{tk}(t) + 1}{6N_{tk}(t)} R(t,i) \tag{18}$$

We define the blocking ratio, $BR(t,i)$, as the ratio of the mean time that a type t transaction blocks other transactions divided by the mean response xecution time of that transaction. That is,

$$BR(t,i) \quad \frac{2N_{tk}(t) + 1}{6N_{tk}(t)}. \tag{19}$$

Observe that $BR(t,i)$ is independent of the site. Consequently, we omit the argument i. observe also that $BR(t)$ is approximately 1/3. This is in agreement with our experimental results where $BR(t)$ was in the range of 0.23 to 0.41. An analysis of blocking by Yu, et al. [YU85] reached similar conclusions.

Let $PB(t,s,i)$ be the conditional probability that a type t transaction at site i is blocked by a type s transaction also at site i, given that a lock request of the type t transaction is blocked. Then, the delay time at the LW delay center for a type t transaction is

$$R_{LW}(t,i) \quad \sum_{s \in T} PB(t,s,i) RLT(s,i). \tag{20}$$

5.5 TM Serialization Delay

The TM server in the CARAT system is equivalent to a critical section in manipulating the shared bookkeeping data structures. The serialization delay due to contention for TM is a factor in transaction response time. However, the processing by the TM server consists only

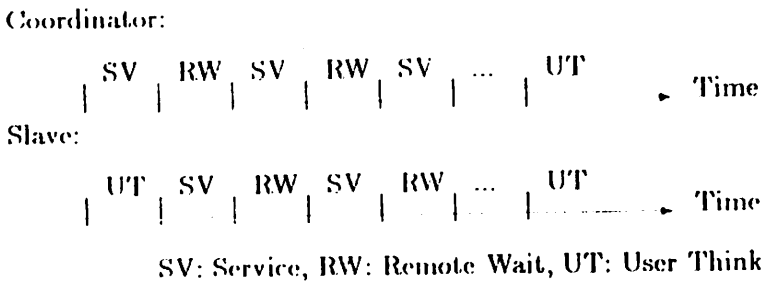

Figure 4: Coordinator remote wait vs. time.

of a burst of CPU time, except at the commit time. To commit a transaction, the TM server at originating site must force-write a commit log record to the log file. If the force-write disk I/O time is small compared to the transaction response time, the net impact of ignoring serialization delay should be very small. Thus, this serialization delay is not considered in the implementation of the solution procedure for the model. However, the reduction technique for analyzing serialization delay [JACO83] can be applied if the serialization delay is to be taken into account.

5.6 Remote Request Wait Delay

The mean remote request wait time, $R_{RW}(t,i)$, for a coordinator transaction of type t at site i, $t \in \{DROC, DUC\}$, corresponds to the mean response time of a remote request plus the round trip network communication delay. The mean response time of a remote request equals the mean slave request response time of the corresponding slave transactions of type s, $s \in \{DROS, DUS\}$, at the slave sites, $S(t)$.

The mean slave request response time can be calculated from $R(s,j)$, the mean response time of the slave transactions of type s at site j, $j \in S(t)$. However, as shown in Figure 4, the contributions of the remote wait time, $D_{RW}(s,j)$, and the inter-transaction time, $D_{UT}(s,j)$, of the slave transaction to $R(s,j)$ should be excluded. That is, the slave request response time consists only of the service times at the CPU, Disk, and LW service centers. Since the number of the remote requests issued by the coordinator transaction between two successive commits is $N_s(t,i) \cdot r(t)$, we have,

$$R_{RW}(DROC,i) \quad 2 \cdot o + \frac{\sum_{j \in S(t)} |R(DROS,j) \quad D_{RW}(DROS,j) \quad D_{UT}(DROS,j)|}{N_s(DROC,i) \cdot r(DROC)}, \quad (21)$$

and,

$$R_{RW}(DUC,i) = 2\cdot\alpha + \frac{\sum_{j\neq s(i)}|R(DUS,j) - D_{RW}(DUS,j) - D_{UT}(DUS,j)|}{N_s(DUC,i)\cdot r(DUC)}, \quad (22)$$

where α denotes the communication delay. The mean remote request wait time, $R_{RW}(s,j)$, for a slave transaction at site j, $s \in \{DROS, DUS\}$, corresponds to the mean elapsed time between the end of a remote request to site j and the initiation of a subsequent remote request to site j by its corresponding coordinator transaction of type t at the coordinator site i, $t \in \{DROC, DUC\}$.

For the remainder of this section we introduce the parameter $f(t,i,j)$ as the fraction of remote requests that a distributed transaction at site i makes to site j, $j \neq i$. This must be input to the model. Since a slave transaction is in remote wait state when its coordinator or other slave transactions are serving the user transaction, we can calculate $R_{RW}(s,j)$ by $R(t,i)$, the mean transaction response time of its corresponding coordinator transaction. The total amount of time a slave transaction spends in the remote wait state is $R(t,i)$ excluding the remote wait time the coordinator spends waiting for the slave transaction, i.e., $R(t,i) - D_{RW}(t,i)\cdot f(t,i,j)$. Hence, we have,

$$R_{RW}(DROS,j) = \frac{R(DROC,i) - D_{RW}(DROC,i)\cdot f(DROC,i,j) - D_{UT}(DROC,i)}{N_s(DROS,j)\cdot l(DROS)}; \quad (23)$$

and,

$$R_{RW}(DUS,j) = \frac{R(DUC,i) - D_{RW}(DUC,i)\cdot f(DUC,i,j) - D_{UT}(DUC,i)}{N_s(DUS,j)\cdot l(DUS)}. \quad (24)$$

5.7 Two-phase Commit Delay

The two-phase commit delay time for a transaction of type t, $t \in \{DROC, DUC\}$, consists of two components: the commit processing time at the slave sites and two round trip communication delay for the two-phase commit protocol. For a successful execution, a DROC or DUC transaction waits for all the DROS or DUS transactions at its slave sites for acknowledgments to the PREPARE and COMMIT messages. Since the two-phase commit messages are processed in parallel at the slave sites, the two-phase commit delay time, $R_{CW}(t)$, is the maximum of the differences in the commit processing time at the slave sites and the coordinator site plus the communication delay. The commit processing time includes the CPU and I/O times spent in the TC and TCIO (or TA and TAIO) phases.

6 Model Solution and Validation

The model is solved iteratively using equations 5 through 10 in Section 5.3. Observe that equations 7 through 10 depend on quantities that are function of the performance measures of the model. Consequently we use an iterative procedure to obtain values for the service

requirements of the LW, RW, CW, and UT service centers so that the relations in equations 7 through 10 are satisfied. Each step of the iteration requires that the site model for each site be solved. This is done using the Mean Value Analysis algorithm for product form networks.

In order to validate the model against empirical measurements, we conducted a series of experiments with the CARAT testbed to obtain performance data for different workloads. From these experiments we obtained the basic parameter values needed for the model, parameters that are independent of the specific workload, as well as empirical values for system performance measures that the model is expected to predict. We first discuss the basic parameter values and then the system performance measures for the four multi-user workloads (LB8, MB4, MB8, and UB6) introduced in Section 2.

As shown in Table 2, there are six basic parameters for each transaction type t at each site (node) i. Other parameter values used by the model are derived from these basic parameters. Each of the parameters corresponds to the CPU or disk I/O requirements for a transaction phase, such as user application processing (U), TM processing (TM), lock request processing (LR), DM disk I/O (DMDIO), etc. In addition to these values, we also measured the CPU times required for the TM server to receive, service, and then send a local or remote message. The additional CPU times required to send or receive network messages for the distributed read (DR) and distributed write (DW) transactions are reflected in $R_{TM}^{(cpu)}$, the mean CPU requirements for the TM phase for that transaction type. Therefore, $R_{TM}^{(cpu)}$ for the DRO and DU transactions are larger than those of the local read-only (LRO) and local update (LU) transactions. Note that the mean CPU time for a lock request, $R_{LR}^{(cpu)}$, includes the CPU cost for local deadlock detection and the user think time, R_{UT}, is set to be zero. Also note that $R_{DMIO}^{(disk)}$, the disk I/O requirements to read or update a database record for a transaction in the DMIO phase, for the LU and DU transactions are three times larger than $R_{DMIO}^{(disk)}$ for the LRO and DRO transactions, because three disk I/O operations, (one read operation to the database file, one write operation to the journal file, and one write operation to the database file) are needed to update a database record, while only one read operation is needed to read a database record. Since only two active nodes were used in the experiments, it turned out that the average Ethernet communication delay, α, was relatively small and therefore could be neglected in the computation. Based on these basic parameters, the resource requirements for each transaction phase was calculated [JENQ86].

Other important parameters that were calculated using the equations discussed in this paper are the blocking probability, $Pb(t,i)$, the deadlock probability, $Pd(t,i)$, and the mean blocking ratio, $BR(t)$. The mean blocking ratio is an important parameter for the estimation of the mean blocking time of a blocked lock request. Based on our analysis we used $BR(t) = 1/3$. This value is in agreement with the results from measurements.

We now present the modeling results and compare the data against the performance measures obtained in the experiments. The number of requests for per transaction, n, was varied from 4 to 20 for each workload to investigate the sensitivity of the results to the length of the transactions. Longer transactions encounter a higher probability deadlock. We use TR-XPUT and Total-CPU refer to the total transaction throughput and the total CPU utilization, respectively, at

Node	t	$R_U^{(cpu)}(t,i)$	$R_{TM}^{(cpu)}(t,i)$	$R_{DM}^{(cpu)}(t,i)$	$R_{LR}^{(cpu)}(t,i)$	$R_{DMIO}^{(cpu)}(t,i)$	$R_{DMIO}^{(disk)}(t,i)$
A	LRO	7.8	8.0	5.4	2.2	1.5	28.0
	LU	7.8	8.0	8.6	2.2	2.5	84.0
	DRO	7.8	12.0	5.4	2.2	1.5	28.0
	DU	7.8	12.0	8.6	2.2	2.5	84.0
B	LRO	7.8	8.0	5.4	2.2	1.5	40.0
	LU	7.8	8.0	8.6	2.2	2.5	120.0
	DRO	7.8	12.0	5.4	2.2	1.5	40.0
	DU	7.8	12.0	8.6	2.2	2.5	120.0

Table 2: Basic Parameter Values (milliseconds).

Node_A or Node_B. The Total-DIO for the measured data denotes the total disk I/O rate. The unit of disk I/O was the database granule, a disk page of 512 bytes. The disk I/O rates for the model were calculated from the the disk utilization, predicted by the model, and the disk service rates. Figures 5 through 10 compare the model predictions with measurements for the LB8 and MB4 workloads. Because the results are displayed for different transaction sizes, n, the transaction throughput data has been normalized by multiplying the transaction completion rate (transactions/second) by the transaction size (database records accessed/transaction) to give a transaction rate in database records/second. Notice that the normalized transaction throughput decreases as the transaction size, n, increases beyond $n = 8$. This is due to an increase in data conflict and rollback caused by deadlocks. The probability that a transaction deadlocks increases rapidly with n. Similar data is given in tabular form for the MB8 and UB6 workloads in Tables 3 and 4. Furthermore, in order to validate the model for each transaction type, we also compared the modeling results for each type against the measurement data in the MB4 workload, as shown in Table 5.

The modeling results agree quite well with the measured data for all the workloads and for each transaction type. Note, however, that the maximum deviation generally occurs in the cases with the smallest transaction size, i.e., when $n = 4$. This can be explained by the fact that, in our implementation of the solution procedure for the model, the TM serialization delay was ignored. For the small transaction sizes, the rate at which messages are sent to the TM server increases and the effects of synchronous disk writes (force-writes by the TM server process to the recovery log) in the two-phase commit execution is more significant. Because the TM serialization delay is significant, the modeled disk I/O rates, and thus, the transaction throughputs, are higher in the model than in the real system.

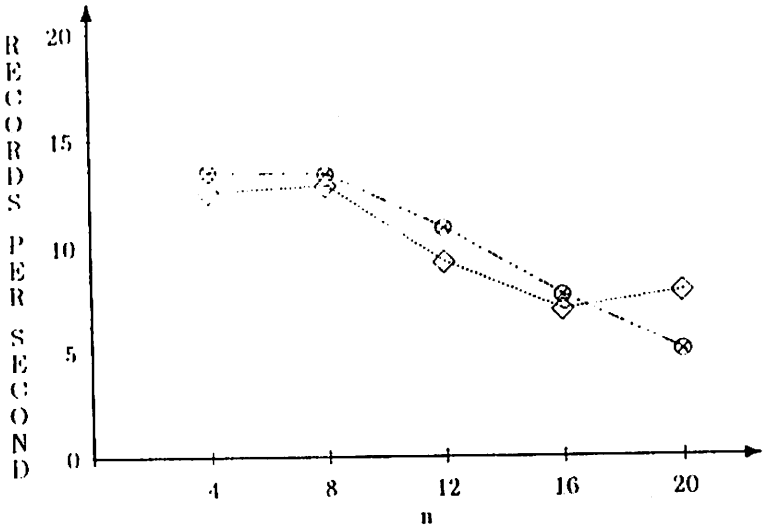

Figure 5: LB8 Workload: Record Throughput (Node B).

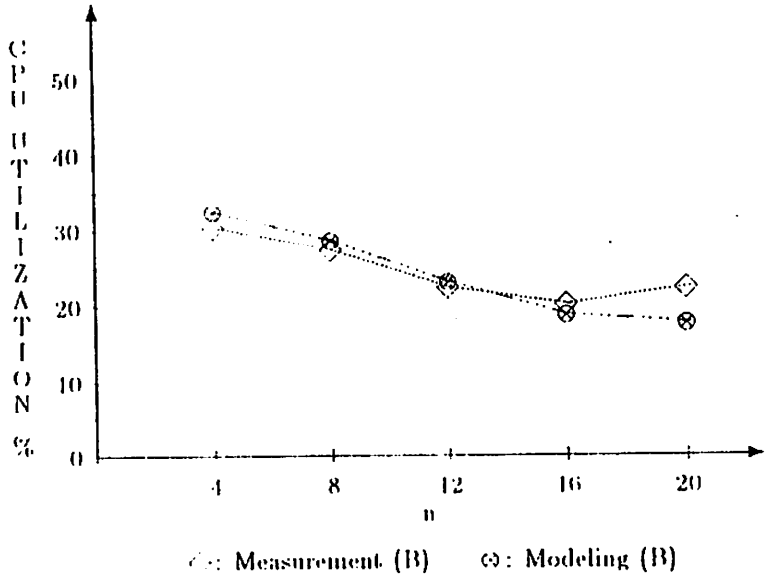

Figure 6: LB8 Workload: CPU Utilization (Node B).

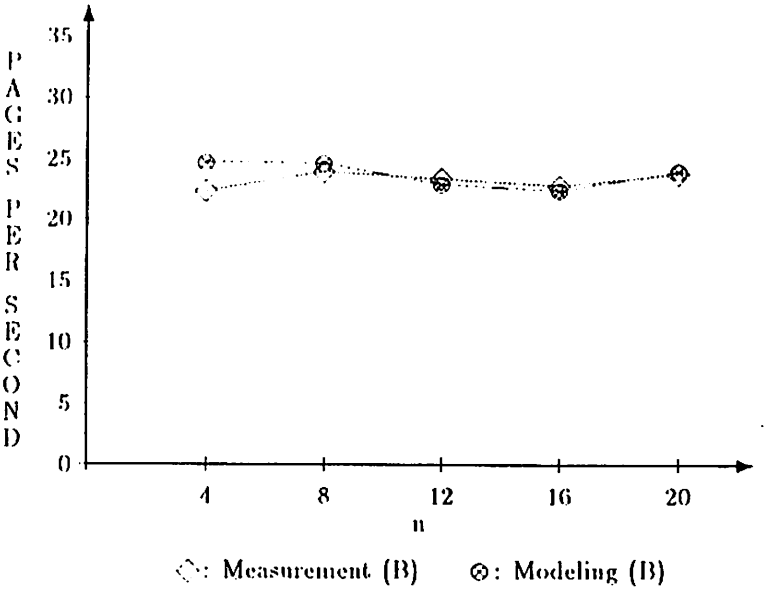

Figure 7: LB8 Workload: Disk I/O Rate (Node B).

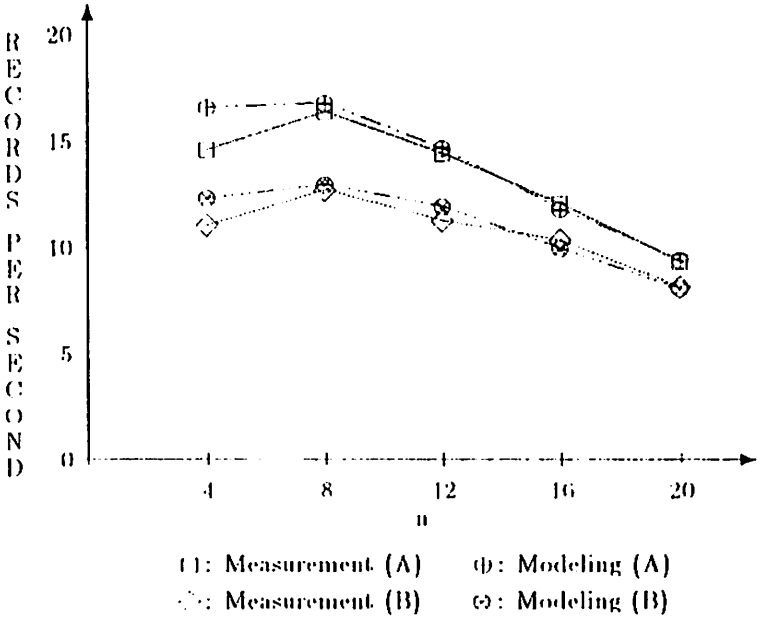

Figure 8: MB4 Workload: Record Throughput.

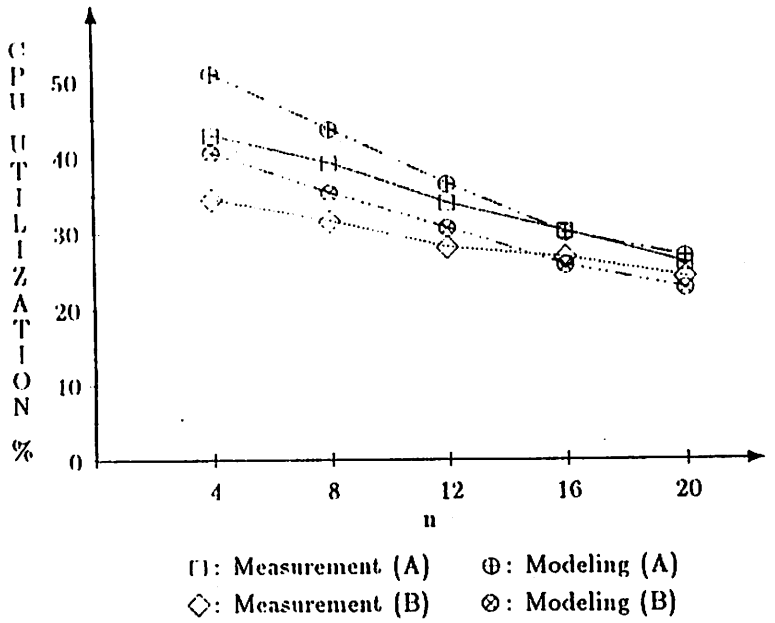

Figure 9: MB4 Workload: CPU Utilization.

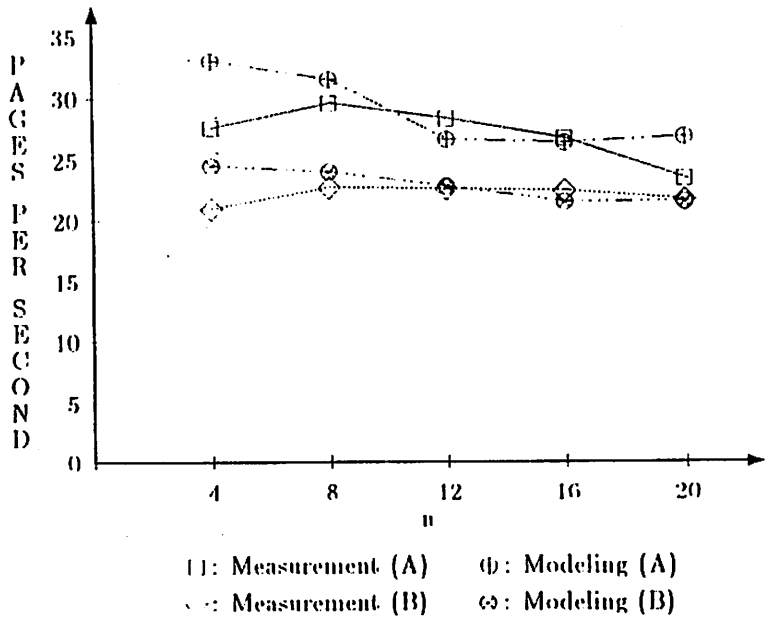

Figure 10: MB4 Workload: Disk I/O Rate.

Test	n	Node	Measurement TR XPUT	Measurement Total CPU	Measurement Total DIO	Modeling TR XPUT	Modeling Total CPU	Modeling Total DIO
MB8	4	A	0.94	0.45	28.9	1.11	0.55	35.1
		B	0.72	0.36	21.9	0.79	0.42	25.0
	8	A	0.45	0.36	28.1	0.54	0.45	32.8
		B	0.39	0.32	23.2	0.41	0.36	24.6
	12	A	0.23	0.31	26.3	0.27	0.33	27.5
		B	0.21	0.27	22.5	0.23	0.29	22.6
	16	A	0.15	0.26	23.4	0.14	0.26	25.6
		B	0.12	0.25	23.0	0.13	0.23	21.4
	20	A	0.09	0.27	23.9	0.09	0.27	30.8
		B	0.08	0.26	23.8	0.08	0.22	23.6

Table 3: Model vs. Measurement Results (MB8).

Test	n	Node	Measurement TR XPUT	Measurement Total CPU	Measurement Total DIO	Modeling TR XPUT	Modeling Total CPU	Modeling Total DIO
UB6	4	A	0.99	0.44	29.6	1.13	0.51	35.1
		B	0.70	0.33	20.9	0.81	0.39	24.9
	8	A	0.53	0.38	30.9	0.56	0.44	33.7
		B	0.39	0.30	23.2	0.42	0.34	24.6
	12	A	0.27	0.31	28.2	0.32	0.35	30.2
		B	0.21	0.25	22.7	0.24	0.28	23.1
	16	A	0.15	0.27	27.0	0.17	0.28	27.9
		B	0.14	0.23	22.0	0.14	0.23	21.8
	20	A	0.10	0.25	24.9	0.10	0.26	30.2
		B	0.08	0.22	21.3	0.08	0.21	22.8

Table 4: Model vs. Measurement Results (UB6).

XPUT		Measurement		Model	
n	Type	Node A	Node B	Node A	Node B
4	LRO	0.39	0.25	0.46	0.29
	LU	0.19	0.11	0.21	0.12
	DRO	0.22	0.22	0.25	0.25
	DU	0.11	0.11	0.11	0.11
8	LRO	0.20	0.13	0.22	0.14
	LU	0.10	0.07	0.11	0.06
	DRO	0.14	0.14	0.14	0.14
	DU	0.07	0.06	0.06	0.06
12	LRO	0.11	0.08	0.12	0.08
	LU	0.06	0.04	0.06	0.04
	DRO	0.09	0.08	0.09	0.09
	DU	0.04	0.03	0.04	0.04
16	LRO	0.07	0.05	0.07	0.05
	LU	0.04	0.03	0.03	0.02
	DRO	0.05	0.07	0.06	0.06
	DU	0.03	0.02	0.03	0.03
20	LRO	0.05	0.04	0.04	0.03
	LU	0.02	0.02	0.01	0.01
	DRO	0.04	0.04	0.04	0.04
	DU	0.02	0.01	0.02	0.02

Table 5: Model vs. Measurement Throughput Results for Each TR Type (MB4).

7 Conclusions

We have developed a queueing network model to analyze the performance of a distributed database testbed system. The model includes the effects of the concurrency control protocol (two-phase locking with distributed deadlock detection), the transaction recovery protocol (write-ahead-logging of before-images), and the commit protocol (centralized two-phase commit). The model has been validated against empirical measurements.

The study differs from other studies in that it presents a model for a functioning distributed transaction processing system. The model must consider the effects of the transaction workload, the two-phase locking scheme, the recovery and journaling costs, etc..

We have validated the queueing network model for a variety of workloads using the data obtained from performance measurements. Note, however, that the approximation of $Pd(t,i)$, the probability of local and global deadlocks, did not include deadlock cycles of length greater than two. Even though deadlock cycles of length greater than two are relatively rare compared to two-cycle deadlocks, it is not insignificant as the transaction size gets larger and the multiprogramming level goes up. Therefore, it would be useful to derive a more accurate model for $Pd(t,i)$ in a multi-node distributed transaction processing environment.

This modeling study does not answer all the important questions, but it establishes a framework for further modeling studies. In order to make practical use of this model in a more general environment than we have investigated, the model should be extended to account for such things as multiple parallel requests, nonuniform and nonrandom database access patterns, the effects of database buffering, and more complex transaction behavior.

Acknowledgments. The autors would like to acknowledge and thank the referees for their careful reading of the original manuscript and their suggestions for improving the presentation.

References

[ALME79] G. Almes and E. Lazowska, "The Behavior of Ethernet-like Computer Communication Network," *Proceedings 7th Symposium on Operating System Principles*, 1979.

[AGRA85a] R. Agrawal, M. J. Carey, and M. Livny, "Models for Studying Concurrency Control Performance: Alternatives and Implications," *ACM SIGMOD International Conference on Management of Data*, 1985, pp. 108-121.

[AGRA85b] R. Agrawal and D. J. DeWitt, "Integrated Concurrency Control and Recovery Mechanisms: Design and Performance Evaluation," *ACM Transactions on Database Systems*, Vol. 10, No. 4, December 1985, pp. 529-564.

[BASK75] F. Baskett, K.M. Chandy, R.R. Muntz, F.G. Palacios, "Open, Closed, and Mixed Networks of Queues with Different Classes of Customers," *J. ACM*, Vol. 22, No. 2, April 1975.

[BERN82] P. Bernstein and N. Goodman, "A Sophisticate's Introduction to Distributed Database Concurrency Control," Research Report, TR-19-82, Harvard University, also *8th Intl. Conference on Very Large Data Bases*, September, 1982.

[CARE84] M. Carey and M. Stonebraker, "The Performance of Concurrency Control Algorithms for Database Management Systems," *Tenth International Conference on Very Large Data Bases*, August 1984.

[CERI84] S. Ceri and G. Pelagatti, *Distributed Databases: Principles and Systems*, McGraw-Hill, New York, 1984.

[CHAN80] K. M. Chandy and C. H. Sauer, "Computational Algorithms for Product Form Queueing Networks," *Communications of the ACM*, October 1980.

[CHAN83] K. M. Chandy, J. Misra and L. M. Haas, "Distributed Deadlock Detection", *ACM Transactions on Computer Systems*, Vol. 1, May 1983, pp. 144-156.

[CHES83] A. Chesnais, E. Gelenbe and I. Mitriani, "On the Modelling of Parallel Access to Shared Data," *Communications of the ACM*, Vol. 26, No. 3, March 1983, pp. 196-202.

[CORN86] D. W. Cornell, D. M. Dias, P. S. Yu, "On Multisystem Coupling Through Function Request Shipping," IEEE Trans. on Software Engineering, Vol. SE-12, No. 10, October 1986, pp. 1006-1017.

[FRAN85] P. Franaszek and J. T. Robinson, "Limitations of Concurrency in Transaction Processing," *ACM Transactions on Database Systems*, Vol. 10, No. 1, March 1985, pp. 1-28.

[GALL82] B. Galler, "Concurrency Control Performance Issues," Ph.D. dissertation, University of Toronto, September 1982.

[GARC79] H. Garcia-Molina, "Performance of Update Algorithms for Replicated Data in a Distributed Database," Ph.D. Dissertation, Computer Science Department, Stanford University, 1979.

[GRAY79] J. N. Gray, "Notes on Data Base Operating Systems," in *Operating Systems: An Advanced Course*, R. Bayer, R. M. Graham, and G. Seegmuller, Editors, Springer - Verlag, 1979, pp. 393-481.

[GRAY81] J. N. Gray, P. Homan, R. Obermack, and H. Korth, "A Straw Man Analysis of Probability of Waiting and Deadlock," IBM Research Report RJ 3066, 1981.

[IRAN79] K. B. Irani and H. Lin, "Queueing Network Models for Concurrent Transaction Processing in a Database System," *ACM SIGMOD International Conference on Management of Data*, 1979, pp. 134-142.

[JACO83] P. Jacobson and E. Lazowska, "A Reduction Technique for Evaluating Queueing Networks with Serialization Delays," *Proceedings of IFIP W.G.7.3 International Symposium on Computer Performance Modeling, Measurement, and Evaluation*, 1983.

[JENQ86] B. P. Jenq, "Performance Measurement, Modelling, and Evaluation of Integrated Concurrency Control and Recovery Algorithms in Distributed Database Systems," Ph. D. Dissertation, Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, February 1986.

[KLEI75] L. Kleinrock, *Queueing Systems - Volume I*, John Wiley and Sons, 1975.

[KOHL86a] W. H. Kohler and B. P. Jenq, "Performance Evaluation of Integrated Concurrency Control and Recovery Algorithms Using a Distributed Transaction Processing Testbed," *Sixth International Conference on Distributed Computing Systems*, Cambridge, MA, May 1986, pp. 130-139.

[KOHL86b] W. H. Kohler and B. P. Jenq, "CARAT: a Testbed for the Performance Evaluation of Distributed Database Systems," *1986 Fall Joint Computer Conference*, November 1986, to appear.

[KRON86] N. P. Kronenberg, H. M. Levy, W. D. Strecker, and R. J. Merewood, "VAXclusters: A Closely-Coupled Distributed System," *ACM Transactions on Computer Systems*, Vol. 4, No. 2, May 1986.

[KUNG81] H. T. Kung and J. T. Robinson, "On Optimistic Methods for Concurrency Control," *ACM Transaction on Database Systems*, Vol. 6, No. 2, June 1981.

[MENA82] D. Menasce and T. Nakanishi, "Optimistic Versus Pessimistic Concurrency Control Mechanisms in Database Management Systems," *Information Systems*, Vol. 7, No. 1, 1982.

[NAKA82] T. Nakanishi and D. Menasce, "Performance Evaluation of a Two-Phase Commit Based Protocol for DDBs," *ACM Principles of Database Systems*, March 1982.

[POTI80] D. Potier and P. Leblanc, "Analysis of Locking Policies in Database Management Systems," *Communications of the ACM*, October 1980.

[RIES77] D. Ries, "Effects of Locking Granularity in a Database Management System," *ACM Transaction on Database System*, September, 1977.

[RIES79a] D. Ries, "The Effects of Concurrency Control on the Performance of a Distributed Data Management System," *Proceedings of 4th Berkeley Workshop on Distributed Data Management and Computer Network*, 1979.

[RIES79b] D. R. Ries and M. R. Stonebraker, "Locking Granularity Revisited," *ACM Transaction on Database System*, June 1979.

[STON83] M. Stonebraker, et al., "Performance Analysis of Distributed Data Base Systems," *Proceedings Third Symp. on Reliability in Distributed Software and Database Systems*, October 1983, pp. 135-138.

[TAY85] Y. C. Tay, N. Goodman and R. Suri, "Locking Performance in Centralized Databases," *ACM Transactions on Database Systems*, Vol. 10, No. 4, December 1985, pp. 415-462.

[THOM82] A. Thomasian, "An Iterative Solution to the Queueing Network Model of a DBMS with Dynamic Locking," *13th Computer Measurement Group Conference*, December 1982, pp. 252-261.

[THOM85] A. Thomasian, "Performance Evaluation of Centralized Databases with Static Locking," *IEEE Transaction on Software Engineering*, Vol. SE-11, No. 4, April 1985.

[YAO77] S. B. Yao, "Approximating Block Accesses In Database Organizations," *Communications of the ACM*, April 1977.

[YU85] P. S. Yu, D. M. Dias, J. T. Robinson, B. R. Iyer, and D. Cornell, "Modeling of Centralized Concurrency Control in a Multi-system Environment," *Performance Evaluation Review*, Vol. 13, No. 2, (Proc. 1985 ACM SIGMETRICS Conference), pp. 183-191.