Analysis of Replication in Distributed Database Systems

BRUNO CICIANI, DANIEL M. DIAS, MEMBER, IEEE, AND PHILIP S. YU, SENIOR MEMBER, IEEE

Abstract—**Geographically distributed database systems have received growing interest in recent years. In this paper, we develop an approximate analytical model to study the tradeoffs of replicating data in a distributed database environment. Several concurrency control protocols are considered including pessimistic, optimistic, and semi-optimistic protocols. The approximate analysis captures the effect of the protocol on hardware resource contention and data contention. The accuracy of the approximation is validated through detailed simulations. We find that the benefit of replicating data and the optimal number of replicates are sensitive to the concurrency control protocol. Under the optimistic and semi-optimistic protocols, replications can significantly improve response time with an additional MIPS requirement to maintain consistency among the replicates. The optimal degree of replication is further affected by the transaction mix (e.g., the fraction of read-only transactions), the communications delay and overhead, the number of distributed sites, and the available MIPS. Sensitivity analyses have been carried out to examine how the optimal degree of replication changes with respect to these factors.**

Index Terms—**Concurrency control, data replication, distributed databases, performance analysis.**

I. INTRODUCTION

IN RECENT years, there has been considerable interest in geographically distributed database systems, in which the databases are distributed among regional processing systems, and some "request shipping" mechanism is provided to support the access of nonlocal data [8], [17], [19], [29], [31]. Since the access of remote data is expensive in terms of the communications overheads and delays involved, data can be replicated, either fully or partially, so that some remote access can be avoided for read-only transactions. However, the overhead for maintaining consistency of replicated copies in the presence of updates can become large. In this paper, we analyze the performance tradeoffs of data replication in distributed database systems. We study the optimal number of replications and the effect of system parameters and the concurrency control protocol on this optimum. Data replication can also improve system availability by providing some tolerance to site and/or link failures. This aspect of data replication is not addressed in this paper.

Previous analytical models of distributed databases with replication have been for very simple models. The models in [7] and [21] are for a single arrival stream of read or update requests, and m parallel servers. There can be up to m simultaneous active read requests, or one active write request with first in first out service. Any write operation in progress blocks all reads behind it, and a write request must wait for any read requests in progress to complete. Using this model for a distributed database system implies that the entire database is locked by any read or write operation. Interpreting each parallel server as a node in a distributed system also implies that communications overheads and delays are ignored. In [25], the model is generalized to capture some contention for physical resources. However, this model also assumes that the entire database is locked for doing updates, and the model is for full replication only. As the authors point out, an extension of their model for incorporating "partial locking" (i.e., not locking the entire database on updates), requires a totally new and difficult analysis. We address these shortcomings of previous models in this paper. Most other previous analyses of concurrency control in distributed database systems have either assumed no data replication [1], [20] or full replications [15], [13]. Simulation studies of partially replicated systems are reported in [28], [5], [2]. In these simulation studies, little variation of the parameter space is considered (presumably because of long simulation times) leading to the incorrect conclusion that in general replication has a negative effect on performance due to update costs.

The purpose of this paper is to examine the performance tradeoffs of data replication in distributed database systems, in terms of the communications overhead and delay, response time constraints, MIPS available, fraction of read-only transactions, the number of sites, and the concurrency control protocol. A secondary objective is to show that a relatively simple approximate analysis method based on a mean value model can estimate system performance with good accuracy. Section II outlines the concurrency control protocols considered. The analytical model is described in Section III. Performance projections from the analytical model and comparison to simulation results are described in Section IV. Concluding remarks appear in Section V.

Manuscript received November 8, 1988; revised October 16, 1989.

B. Ciciani is with the Departiment di Ingegneria Elettronica, Seconda Universite di Roma, Rome Italy. This work was done while he was a visiting scientist at the IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598.

D. M. Dias and P. S. Yu are with the IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598.

IEEE Log Number 9035103.

II. Concurrency Control Protocols

We assume that a number of geographically distributed computer systems (sites) $S_1 \cdots S_n$ are interconnected by a (long haul) computer network. The (global) database D is partitioned among the sites, with site S_k, $k \in \{1, \cdots, n\}$, being the master (primary) site for partition D_k (as detailed below). Essentially, for any data accessed from partition D_k by a transaction, site S_k must be consulted before the transaction can be committed. (A transaction is said to be committed when its effects are made permanent and the data it writes become visible to other committing transactions.) Each partition D_k is replicated at m additional sites; $m = 0$ implies no replication and $m = (n - 1)$ implies full replication. Intermediate values of m result in partial replication. For ease of presentation, we will consider the homogeneous case where each partition D_k, $k \in \{1, \cdots, n\}$, has M/n database granules. The analysis is readily generalized to the nonhomogeneous case with arbitrary numbers of granules at each site.

The following classes of transactions are considered: class 1 transactions entering at site k, only access data belonging to partition D_k. Class 2 transactions access some data belonging to some other partition i, $i \neq k$. In all of the protocols, if a class 2 transaction needs access to a granule for which there is a local replication, the access is made locally (perhaps with some communication for concurrency control as detailed below). If the data granule is not available locally, a remote data access is made from the master (primary) site for the partition containing the granule. At transaction commit time, updates are propagated to all replicates of the data. In general, a transaction reads one copy, and writes all copies of the data. The concurrency control protocols ensure that transactions access consistent data in this environment.

We consider two basic types of distributed concurrency control protocols: a distributed optimistic protocol (OPT) (distributed certification in the terminology of [3] and [4]) and a distributed two-phase locking (2PL) protocol. We also refer to the latter as the pessimistic protocol. To handle data replication, both these protocols use the primary copy approach [4] as detailed below. A 2PL protocol with primary copy is used by INGRES [24]. Finally, we consider a combination of the two in which class 1 transactions use 2PL (since all accesses are at the primary site) and class 2 transactions use the OPT protocol. Other approaches to handling replication such as the "do nothing" approach in [4] could be analyzed in a similar manner, but are not considered here.

The distributed optimistic protocol we consider is based on the use of weak locks, and is as follows. Each site has a concurrency controller (CC) that accepts requests for weak or strong locks. A weak lock request is always granted if no conflicting strong locks are held on the data; otherwise, the weak lock request is rejected. A strong lock request is granted if it does not conflict with any concurrently held strong lock; otherwise, it is rejected. Further, if any conflicting weak locks are currently held, the holding transactions are marked for abort and the conflicting weak locks are released. An incoming transaction is run at the site of origin. For each data granule accessed during the transaction a weak lock is requested at the site at which the data are obtained (i.e., if a local copy of the data granule is available, the data are accessed locally and a local weak lock is requested; otherwise, the data are accessed at the primary site where the weak lock is also requested). If the weak lock request is rejected, the transaction is marked for abort. No updates are reflected into any replicates until the end of the transaction. At that time, a two-phase commit protocol [11] is used as follows. If the transaction has been marked for abort it releases its local weak locks, waits for a backoff period and restarts; otherwise, a "prepare to commit" message is sent to all the master sites of the data accessed, along with the identification of the granules involved and the access mode. At the master sites, an attempt is made to obtain strong locks on the data. If any strong lock request is rejected, the master sites return a not certified (NAK) to the requesting site; otherwise, a positive acknowledgment (ACK) is returned. (Note that if a strong lock request is rejected, the transaction is not placed in a wait queue, but is aborted.) If a NAK is received from any of the master sites, the transaction is aborted and messages are sent to the master sites to release any strong locks acquired. If all responses are ACKs, the second phase of commit processing begins and commit messages are sent to the master sites along with any update information. The primary site sends the update information to all the sites holding copies of the data; if a weak lock is held at any of these sites, then the corresponding transactions are marked for abort. When the master sites get an acknowledgment that the data have been updated at all replicates, the strong locks on the data are released and an ACK is sent back to the originating site, completing the commit process.

This optimistic protocol is similar to the 2PL certifier outlined in [4] except that in the above protocol a conflict detected during certification (commit process) causes the conflicting active transaction to abort; the reverse is done in [4]. Intuitively, it is preferable to abort an ongoing transaction and commit a transaction in certification rather than the converse since there is no guarantee that an ongoing transaction will successfully commit. A time-stamp-based optimistic method (certification) is described [23] and simulated in [5]. Under both the weak lock method described here and the time-stamp based certifier in [23], a transaction will be aborted if it reads a data item which is different from the most recent version produced by already committed transactions. The difference between the protocols is largely in terms of the implementation for resolving conflict between transactions that are simultaneously in commit, resulting in similar behavior in terms of communications overhead and aborts.

We compare the above optimistic protocol with the 2PL protocol using the primary copy approach to handle replication [4]. To ensure serializability, locks are obtained at the primary site. A single phase of commit suffices to update all the replicates and then release the locks. Global deadlocks can occur and can be detected using the various

methods [24]. In the analysis, we neglect the probability of deadlock. The justification is that both our simulations and estimates in [27] and [16] show that the deadlock probability is negligibly small compared to the contention probability (which we model in detail).

Finally, we consider a protocol that uses 2PL for class 1 transactions, and OPT for class 2 transactions. We refer to this as the semi-optimistic (SEMI) protocol. Committing class 2 transactions now find strong locks held by active class 1 transactions, and therefore abort. Class 1 transactions are never aborted. This is desirable if transactions accessing local data only are to be given priority over class 2 transactions.

In the above protocols, a transaction is not committed until updates are reflected in all replicates. We refer to this as the synchronous version of the protocols. It offers the advantage that in case of the failure of a primary copy site, a new primary site can be designated for the partition, and an up-to-date version of the partition can be created at that site. We compare this to the same protocols with asynchronous updates to the replicates, where the transaction is committed after only the primary site is updated. Replicates are then updated from the primary copy site asynchronously. Consistency in this environment can be guaranteed by a count mechanism at the primary site [6]. The time-stamp mechanism of [23] will also permit asynchronous updates (because the time stamp of the data at the replicate and primary site will be different until the asynchronous update is completed). The asynchronous update allows better performance with loss of availability of the partition controlled by a failing site until recovery is completed at that site. We quantify this performance difference.

III. The Model

In this section, we develop an approximate analytical model to estimate and compare the performance of the protocols. A simulation model is used to validate the analysis methodology. The model partitions the analysis into two interacting components: resource contention and data contention. Queueing models are used to estimate the hardware resource contention effects, and are detailed in Sections III-B and III-C. The estimation of the lock contention wait time and the contention and abort probabilities are presented in Sections III-D and III-E. Notice that for the optimistic and the semi-optimistic protocols, data contention between class 1 and class 2 transactions is manifested as an abort, leading to resource consumption for a rerun transaction, and consequently higher resource contention, leading in turn to higher data contention. Such interaction between the data and resource contention is estimated using an iteration which is an extension of the methodology developed in [32] and [33]. The approach converges in a few iterations.

We describe in detail the analysis of the semi-optimistic protocol with asynchronous updates, since this protocol involves both the elements of 2PL (for class 1 transactions) and of optimistic concurrency control (for class 2 transactions). The analysis of the other protocols can be derived based on a similar approach. The methodology described below also analyzes processor utilizations, transaction abort, and contention probabilities as a function of transaction rate.

A. Overview

The methodology and analysis method extends that for centralized database systems using locking reported in [30] and [32], distributed databases without replication also using locking [12], and hybrid distributed-centralized systems with single replication [6], to fully distributed systems with partial replication and with different concurrency control mechanisms. This architecture has a more complicated concurrency control method with a combination of optimistic and pessimistic concurrency control, and includes communications overheads and delays that impact contention. The approximation method has a similar flavor to the mean value analysis in [26] for locking in centralized databases, using the steady-state average values of variables rather than probability distributions of the variables. As in [32], we decompose the resource and data contention and capture their interaction using an iteration. In most of the previous work, the interaction between data and resource contention is either ignored or the resource contention is not modeled [18], [22], [16], [14], [26]. Recently, a similar decomposition approach for optimistic concurrency control in centralized systems has been reported in [33]. (A slightly different approach using a piecewise linear model to capture resource contention is used in [9].) In [34], a class of hybrid optimistic concurrency control schemes which switch from the optimistic scheme to locking during rerun is described and analyzed using the same approach. Previous models of distributed databases using replication have either used a very simple read/write model and ignored resource contention [7], [21] or have resorted to simulation [2], [5].

Because of the difference in their response times we distinguish three kinds of transactions: 1) class 1 transactions with average response time of R_T^{CLASS1}, 2) first run class 2 transactions with average response time R_{T1}^{CLASS2}, and 3) rerun class 2 transactions with average response time of R_{T2}^{CLASS2}. Weak or strong locks are assumed to be all held until the end of the transaction. For the estimation of contention and abort probabilities, we are leaving out an initial portion of the transaction response time, during which the transaction goes through a transaction setup phase (and no locks are held). A transaction that is rerun after an abort is modeled to find all data referenced in the main memory of the system where the data are accessed. This assumption is valid if the main memory buffer can hold the working set of running transactions; detailed buffer models in [10] show that this is valid except for very small main memory buffers. Here, we let aborted transactions run to completion instead of aborting them during the run so that a rerun transaction will find all data referenced in main memory. This is found to improve performance in [33] since it increases the chance of successful completion during a rerun. Further, locks are released

after an abort. The difference between R_{T1}^{CLASS2} and R_{T2}^{CLASS2} is the I/O time and processing time and wait time for locks.

In Section III-B (III-C), we derive expressions for the class 1 (class 2) transaction average response time in terms of contention and abort probabilities. In Section III-D the average wait times for each lock contention and the contention and abort probabilities are derived. The contention and abort probabilities are derived using essentially the following method. A conflict is defined as an event in which a transaction requests a lock on an item that is currently held by another transaction in an incompatible mode. The conflict rates among class 1 (respectively, class 2) transactions and between class 1 and class 2 transactions are derived. Conflicts among class 1 transactions are mapped into lock waits, while conflicts among class 2 or between class 1 and class 2 transactions are mapped into aborts of the class 2 transaction.

In the performance comparisons of Section IV, the parameters used for the transaction characteristics are similar to those in [8], and are derived from trace analysis in [32]. In the following sections, the parameter values used for both the transaction characteristics and for system overheads are indicated as the parameters are defined.

B. Response Time for Class 1 Transactions

In order to evaluate the class 1 transaction response time, transaction arrivals are modeled by a Poisson process. The average class 1 transaction response time is estimated as

$$R_T^{CLASS1} = R_{CPU}^{CLASS1} + R_{IO}^{CLASS1} + R_{CONT}^{CLASS1}.$$

Each component will be described separately.

R_{CPU}^{CLASS1} is the total time the transaction spends at the CPU. This includes both the CPU service and queueing times. The instructions executed by the transaction are divided into those associated with database calls (ten calls per transaction with 25K instructions per call) and those for transaction initiation and application loading (150K instructions per transaction, denoted as INPL). Lock requests and database calls are assumed to occur uniformly over the transaction execution, breaking the transaction into many small tasks of equal size, each of which has to queue and be serviced by the CPU. From prior trace analysis [32] an average of 15 lock requests per transaction is used. In addition, each lock/unlock request entails additional processing (1K instructions per lock/unlock request is used in the model). A communication overhead of 20K instructions is assumed equally split between sending and receiving a message from one system to another. An overhead of 3K instructions per I/O is used. Finally, updates by the class 1 transaction are asynchronously propagated to all the replicates, involving a communications and update overhead. Note, however, that this overhead increases CPU utilization but is not directly included in the response time of class 1 transactions. CPU service time and queueing time are evaluated by modeling the CPU as an M/M/1 queue. Then R_{CPU}^{CLASS1} is expressed as

$$R_{CPU}^{CLASS1} = R_{CPU.INPL}^{CLASS1} + R_{CPU.PROC}^{CLASS1}$$

where the first term is the component for the initial processing (INPL) when no locks are held, and the second term is associated with the remaining processing.

R_{IO}^{CLASS1} is the amount of time spent during the transaction, waiting for I/O to occur. Note that for each I/O (or remote lock request) the processor is modeled to task switch to process another transaction, while suspending the executing transaction until the completion of the I/O. Thus,

$$R_{IO}^{CLASS1} = t_{IO} n_{IO}$$

where t_{IO} is the average time per I/O. Sufficient I/O bandwidth is assumed to enable the modeling of the I/O server as an infinite server with a load independent service time of 35 ms. n_{IO} is the average number of I/O's per transaction, and consists of two kinds of I/O's. n_{IOPL} is the average number of I/O's per transaction that must occur for a transaction to start processing. Typically, these are the I/O's needed to load the application program and the other constructs into the computer memory from disk. Trace analysis yielded a value of 5 for n_{IOPL}. n_{IODB} is the average number of I/O's that occur during the execution of the transaction. These are required to read and write data from disk resident databases into and from the main memory based database buffer, respectively. In the trace analysis, the average transaction performed 11 database I/O. Then

$$R_{IO}^{CLASS1} = (n_{IOPL} + n_{IODB}) t_{IO} = R_{IOPL}^{CLASS1} + R_{IODB}^{CLASS1}.$$

R_{CONT}^{CLASS1} is the time spent in contention wait for a lock that is held by another class 1 transaction or class 2 transactions during their coherence phase, and is derived in Section III-D.

C. Response Time for Class 2 Transactions

The total response time of a class 2 transaction is expressed as

$$R_T^{CLASS2} = R_{T1}^{CLASS2} + R_{T2}^{CLASS2} p_{A1} + R_{T3}^{CLASS2} p_{A1} \frac{p_{AS}}{1 - p_{AS}} + R_{COMMIT}^{CLASS2}.$$

R_{T1}^{CLASS2} covers the time to run a new class 2 transaction and the second and the third terms account for the average rerun time. R_{COMMIT}^{CLASS2} includes 1) the communications delays from the site of origin of the transaction to the sites that are masters of the data accessed for the authentication and second commit phase, 2) the CPU time at the master sites and site of origin for communication, authentication, and commit phase overhead, and 3) the I/O time and overhead for updating the local database during the second commit phase. At the site of origin, the two-phase commit processing overhead includes a constant overhead for each

phase (2K instructions) plus an overhead for each local system involved in the commit operation (3K instructions per system in each phase), and communications overhead; at each master site involved in the commit there is communications overhead, an overhead for authentication (2K instructions), and an overhead to update the master database. (Asynchronously, the master sites propagate the updates to all replicates, involving a communications and update overhead. Note, however, that this overhead increases CPU utilization but is not directly included in the response time of the class 2 transaction.) p_{A1} is the probability of the first abort of a class 2 transaction, and p_{AS} is the probability of the second and subsequent aborts of class 2 transactions. Analogous to the development of the response time expressions for the class 1 transaction we write

$$R_{T1}^{\text{CLASS2}} = R_{\text{CPU1}}^{\text{CLASS2}} + R_{\text{IOPL}}^{\text{CLASS2}} + R_{\text{IODB}}^{\text{CLASS2}} + R_{\text{COMM}}^{\text{CLASS2}}.$$

The component $R_{\text{CPU1}}^{\text{CLASS2}}$ equals $R_{\text{CPU1.INPL}}^{\text{CLASS2}} + R_{\text{CPU1.PROC}}^{\text{CLASS2}}$ to execute the first run of a class 2 transaction, and is similar to that for the class 1 transaction described above, except that for each reference to data that is not available at the local site (i.e., no local replication of the data) a remote access is made to the site involved, incurring a communications and access overhead. In the latter case, the database call overhead and associated I/O and concurrency control overhead is modeled as occurring on the remote system. As before, an M/M/1 model is used to estimate $R_{\text{CPU1}}^{\text{CLASS2}}$ as well as $R_{\text{COMMIT}}^{\text{CLASS2}}$. $R_{\text{IOPL}}^{\text{CLASS2}}$ and $R_{\text{IODB}}^{\text{CLASS2}}$ for class 2 transactions are similar to those for class 1 transactions. $R_{\text{COMM}}^{\text{CLASS2}}$ includes the communications delay from the site of the origin of the transaction to the remote sites for each nonlocal access to data made during each run of the transaction. The remaining terms to evaluate in the above expression are R_{T2}^{CLASS2} and R_{T3}^{CLASS2}. Recall that in the protocol, class 2 transactions use optimistic concurrency control, and abort in case of a conflict with a class 1 or class 2 transaction. We assume that all the data for rerun transactions are available in buffers at the site involved, and consequently there are no I/O's for the rerun transaction. Thus,

$$R_{T2}^{\text{CLASS2}} = P_{\text{GL1}}(R_{\text{COHER}} + T_{\text{Backoff}}) + R_{\text{CPU2.PROC}}^{\text{CLASS2}} + R_{\text{COMM}}^{\text{CLASS2}}$$

and

$$R_{T3}^{\text{CLASS2}} = P_{\text{GL2}}(R_{\text{COHER}} + T_{\text{Backoff}}) + R_{\text{CPU2.PROC}}^{\text{CLASS2}} + R_{\text{COMM}}^{\text{CLASS2}}$$

where $R_{\text{CPU2.PROC}}^{\text{CLASS2}}$ is the same as $R_{\text{CPU1.PROC}}^{\text{CLASS2}}$ excluding the overheads for I/O. R_{COHER} includes 1) communications delays from the site of origin to the sites that are masters of the data accessed and 2) CPU time for communication and authentication phase overhead. P_{GL1} is the probability that the first abort is detected after communication with an external master site during the authentication phase. Similarly, P_{GL2} is the probability that a subsequent abort is detected after communication with an external master site during the authentication phase. A methodology to derive these probabilities is given below. The time T_{Backoff} is the backoff time for a class 2 transaction that aborts due to contention with a running class 1 transaction during the authentication phase.

D. Contention Wait Time Estimation

$R_{\text{CONT}}^{\text{CLASS1}}$ is the time spent in contention wait by a class 1 transaction for a lock that is held by another transaction. The contention wait is estimated as

$$R_{\text{CONT}}^{\text{CLASS1}} = R_{\text{CONT.CLASS1}}^{\text{CLASS1}} + R_{\text{CONT.COHER.CLASS2}}^{\text{CLASS1}}$$

where $R_{\text{CONT.CLASS1}}^{\text{CLASS1}}$ is the time spent in contention wait for a lock that is held by another local class 1 transaction and $R_{\text{CONT.COHER.CLASS2}}^{\text{CLASS2}}$ is the time spent in contention wait for a lock that is held by a local or external class 2 transaction during its authentication phases and commit phase. We use the following approach to evaluate $R_{\text{CONT.CLASS1}}^{\text{CLASS1}}$.

$$R_{\text{CONT.CLASS1}}^{\text{CLASS1}} = N_L P_{\text{CLASS1.CLASS1}} \overline{W}_{\text{CLASS1}}$$

where N_L is the mean number of locks per transaction, $P_{\text{CLASS1.CLASS1}}$ is the probability of contention on a lock request with a local class 1 transaction, and $\overline{W}_{\text{CLASS1}}$ is the average time a class 1 transaction waits if it contends with another class 1 transaction for a lock. The evaluation of $P_{\text{CLASS1.CLASS1}}$ is given in Section III-E. The wait time $\overline{W}_{\text{CLASS1}}$ is estimated as $\overline{W}_{\text{CLASS1}} = \beta_{\text{CLASS1}}/F$, where $\beta_{\text{CLASS1}} = R_{\text{CPU.PROC}}^{\text{CLASS1}} + R_{\text{IODB}}^{\text{CLASS1}} + R_{\text{CONT}}^{\text{CLASS1}}$ is the period during which locks are held by a class 1 transaction. Assuming that locks are acquired uniformly during the period β_{CLASS1}, the factor F is estimated as 3 in [30]. Similarly, $R_{\text{CONT.COHER.CLASS2}}^{\text{CLASS1}}$ is estimated as

$$R_{\text{CONT.COHER.CLASS2}}^{\text{CLASS1}} = N_L\left(P_{\text{CLASS1.COHER.CLASS2}}\frac{R_{\text{HOLD}}}{2} + P_{\text{CLASS1.COHER.COMMIT.CLASS2}}\frac{R_{\text{HOLD.COMMIT}}}{2}\right)$$

where $P_{\text{CLASS1.COHER.CLASS2}}$ is the probability of contention on a lock request with a class 2 transaction in its authentication phase that is subsequently aborted. $P_{\text{CLASS1.COHER.COMMIT.CLASS2}}$ is the probability of contention on a lock request with a class 2 transaction in its authentication and commit phase that successfully commits. The manner in which $P_{\text{CLASS1.COHER.CLASS2}}$ and $P_{\text{CLASS1.COHER.COMMIT.CLASS2}}$ are projected is described later. R_{HOLD} is the amount of time a class 2 transaction holds locks at the cohort sites during its authentication phases, when it is subsequently aborted, while $R_{\text{HOLD.COMMIT}}$ is the amount of time a class 2 transaction holds locks at the cohort sites during its last authentication and commit phase. These are the times spent in the

CPU for authentication and committing estimated from an M/M/1 model plus the communication delays. Since all locks are obtained at the beginning of the authentication phase, the average wait time for such contentions are estimated as $R_{\text{HOLD}}/2$ and $R_{\text{HOLD.COMMIT}}/2$, respectively, in the above equations.

E. Contention and Abort Probabilities

We now derive the class 1 lock contention and class 2 transaction abort probabilities. We define a conflict as an event in which a transaction requests a lock that is currently held by another transaction in an incompatible mode. Conflicts among class 1 transactions are manifested as lock contention wait for the element to be unlocked by the other transaction holding the lock. Conflicts between class 1 and class 2 transactions (not in authentication/commit phase) manifest themselves as aborts of the class 2 transactions.

Each local system has a local database and is subject to a transaction rate λ. Of these transactions, a fraction P_{CLASS1} are class 1 and a fraction P_{CLASS2} are class 2. We estimate

$$P_{\text{CLASS1.CLASS1}} = \frac{\lambda P_{\text{CLASS1}} N_L \dfrac{\beta_{\text{CLASS1}}}{2}}{\text{LSPACE}/N_{\text{DS}}}.$$

In this equation, the numerator is the average number of locks held by class 1 transactions, estimated from Little's law as the product of the average number of locks requested by class 1 transactions per unit time and the average lock holding time. LSPACE is the number of database granules across all the N_{DS} distributed sites, and we assume that each distributed system is master for $\text{LSPACE}/N_{\text{DS}}$ database granules. The term $\beta_{\text{CLASS1}}/2$ is the average lock holding time assuming that a transaction acquires locks uniformly over the time that it holds locks.

Similarly, we may write the other conflict probabilities as

$$P_{\text{CLASS1.COHER.CLASS2}} = \frac{\lambda P_{\text{CLASS2}} N_L p_{A1} \left(P_{\text{GL1}} + \dfrac{p_{AS}}{1 - p_{AS}} P_{\text{GL2}} \right) R_{\text{HOLD}}}{\text{LSPACE}/N_{\text{DS}}}$$

$$P_{\text{CLASS1.COHER.COMMIT.CLASS2}} = \frac{\lambda P_{\text{CLASS2}} N_L R_{\text{COHER.COMMIT}}}{\text{LSPACE}/N_{\text{DS}}}.$$

For the parameter values considered, $P_{\text{CLASS1.COHER.CLASS2}}$ is very small, since it includes a product of the form $P_{A1} P_{\text{GL1}}$, and this term is neglected subsequently.

We now estimate the class 2 transaction abort probabilities. The key observation is that class 2 transactions are aborted either by committing class 1, committing class 2 transactions, or because a class 1 or class 2 transaction is holding conflicting locks when a class 2 transaction gets to the primary site during authentication. The probability of abort of a first run class 2 transaction is estimated as

$$\begin{aligned} P_{A1} = \frac{\lambda N_{\text{DS}}}{\text{LSPACE}} \Bigg\{ & P_{\text{CLASS1}} \left(\frac{\beta^{\text{NEW}}_{\text{CLASS2}}}{2} + 2 \times \text{COMDEL} \right) N_L^2 \\ & + P_{\text{CLASS2}} \left(\frac{\beta^{\text{NEW}}_{\text{CLASS2}}}{2} + 2 \times \text{COMDEL} \right) N_L^2 \\ & + P_{\text{CLASS1}} \left(\frac{\beta_{\text{CLASS1}}}{2} \right) N_L^2 \\ & + P_{\text{CLASS2}} (R_{\text{HOLD.COMMIT}}) N_L^2 \Bigg\} \\ = \frac{\lambda N_{\text{DS}}}{\text{LSPACE}} & \times X \end{aligned}$$

where X denotes the term in braces. In this equation, the first term in braces is for the contention with committing class 1 transactions, the second term is due to contention with committing class 2 transactions, the third term (similar to $P_{\text{CLASS1.CLASS1}}$) is for contention of a class 2 transaction arriving at a primary site during authentication with a class 1 transaction that is running, and the last term (similar to $P_{\text{CLASS1.COHER.COMMIT.CLASS2}}$) is for contention of a class 2 transaction arriving at the primary site during authentication with a class 2 transaction holding strong locks at the primary site during its commit process. For instance, $\lambda N_{\text{DS}} P_{\text{CLASS1}}$ is the rate at which class 1 transactions commit; $\beta^{\text{NEW}}_{\text{CLASS2}}/2 = (R^{\text{CLASS2}}_{\text{CPU1.PROC}} + R^{\text{CLASS2}}_{\text{IODB}} + R^{\text{CLASS2}}_{\text{COMM}})/2$ is the average vulnerability period of a first run class 2 transaction, so that any class 1 transaction that commits in this interval causes an abort of the class 2 transaction. Hence, $\lambda N_{\text{DS}} P_{\text{CLASS1}} N_L (\beta^{\text{NEW}}_{\text{CLASS2}}/2)$ estimates the average number of data items invalidated by committing class 1 transactions. Dividing this by LSPACE estimates the probability that a class 2 transaction accesses a data item that is invalidated by a committing class 1 transaction, causing it to abort. Assuming that at most one such contention can occur per class 2 transaction, the transaction abort probability due to this factor is estimated by multiplying the single access conflict probability by N_L. A more accurate estimate can be obtained by estimating the probability of no abort for each access and taking the product of these terms to estimate the probability of no abort for the transaction [33]. The difference between such estimates was found to be negligible. The term of $2 \times \text{COMDEL}$ arises because the vulnerability period of the class 2 transaction is increased by this time due to communications delay in the authentication phase. One factor of COMDEL in this vulnerability period arises because any transaction committing at a remote primary site within a COMDEL before the start of a class 2 transaction may send an update request to abort the transaction. The second COMDEL factor arises because any transaction that commits at a primary site within a COMDEL after the completion of the execution of a class 2 transaction at the originating site can again cause

this transaction to abort. In a similar manner, the other terms in the equation can be derived. The probability of abort P_{AS} of rerun class 2 transactions is estimated by an analogous expression to the above with $\beta^{\text{NEW}}_{\text{CLASS2}}$ replaced by $\beta^{\text{RERUN}}_{\text{CLASS2}} = R^{\text{CLASS2}}_{\text{CPU2.PROC}} + R^{\text{CLASS2}}_{\text{COMM}}$. Finally, recall from Section III-C that the factor P_{GL1} (respectively, P_{GL2}) is defined as the probability that the first (respectively, subsequent) abort of class 2 transactions are detected after communication with other sites during the authentication phase. In the above equation for P_{A1}, the terms involving twice the communications delay are due to aborts detected after communications to the master sites, and similarly the final two terms due to locks held by running class 1 transactions and committing class 2 transactions also involve a communications before abort. Hence, we estimate

$$P_{\text{GL1}} = \frac{P_{\text{CLASS1}}\left(2 \times \text{COMDEL} + \frac{\beta_{\text{CLASS1}}}{2}\right) + P_{\text{CLASS2}}(2 \times \text{COMDEL} + R_{\text{HOLD.COMMIT}})}{X}.$$

P_{GL2} is estimated in a similar manner.

IV. Performance Comparison

In this section, we first compare the results of the analysis in the preceding section to simulation estimates. Following this, we present performance projections based on the approximate analytic model.

A. Simulation

A detailed event driven simulation was developed to simulate the semi-optimistic protocol which combines both the pessimistic protocol (for class 1 transactions) and the optimistic protocol (for class 2 transactions). Simulation results are presented for a ten-site distributed system, with a fully connected network of 200 ms communication delay between sites. Each CPU has 3 MIPS. Transaction arrivals are Poisson processes, with the same arrival rate at each distributed site. The probability of class 1 transactions is chosen as 0.5. Overheads and pathlengths are the same as described in Section III. In the simulation, a global lock space of 32K elements is used. Class 1 transactions make data requests uniformly over the lock space for which the local site is the master, i.e., one tenth of the lock space, while class 2 transactions make lock requests uniformly over the entire lock space. The simulation maintains lock tables and explicitly simulates lock contention, and waits for locked entities, queueing, and processing at the CPU, communications delay and overhead, I/O waits, aborts of central and/or local transactions, and commit processing. The CPU service times are constants equal to the time to execute the specific instruction pathlengths given in Section III-B (and are not exponentially distributed). The CPU is released by a transaction when lock contention occurs, for each I/O, and for communication to another site. In the case of a contention that leads into a deadlock the transaction is aborted and all locks held are released. (Note that for this protocol deadlocks are purely local since locks are only used for contention among class 1 transactions running at the same site.)

We now compare results of the simulation and analysis for the above parameters for the semi-optimistic protocol with asynchronous update. Point estimates from the simulation are the averages for 10 000 transactions for each simulation run, after discarding the statistics of the first 5000 transactions. Confidence intervals are not indicated because the length of the simulation time required made this infeasible. Fig. 1 shows the average response time versus total transaction rate for class 1 and class 2 transactions, for the fully replicated case. The estimates for the simulation and analysis are very close. Fig. 2 shows the same types of curves for the case with only partial replications. Four additional replications are introduced for each database. The simulation and analysis are again very close until the knee where each CPU is close to 95% utilized. As the abort probability strongly affects the system performance, we show the first abort probability of the class 2 transactions in Fig. 3. Close agreement is observed between the simulation and analysis. Note that the simulation is rather time consuming even for this modest configuration, taking about 10 min of CPU time on an IBM 3090 for each point simulated. We present the comparison study in Section IV-B based on the approximate analysis.

B. Performance Study

We now examine the performance tradeoffs between the protocols and the sensitivities to the number of replications, communications overhead and delay, transaction mix, and system configuration. Results are presented for the case of 20 geographically distributed sites, unless otherwise stated. A global lock space of 100K elements is assumed. Unless otherwise specified, the CPU at each site is assumed to be of 10 MIPS, the fraction of class 1 transaction is 75%, communications overhead is 20K, and communications delay is 2.0 s.

We first examine the tradeoff between different protocols, semi-optimistic, pure optimistic, and pessimistic, for different degrees of replications, and with varying communications overhead, when all transactions update the database. Later in the section, we examine the effect of ready-only transactions. The effect of asynchronous versus synchronous update is demonstrated for the semi-optimistic protocol. The other protocols are assumed to use the asynchronous update policy. Fig. 4 shows the average transaction reponse time versus the number of replications of each partition for the case of zero (negligible) communications delay. The cases of full and no replications

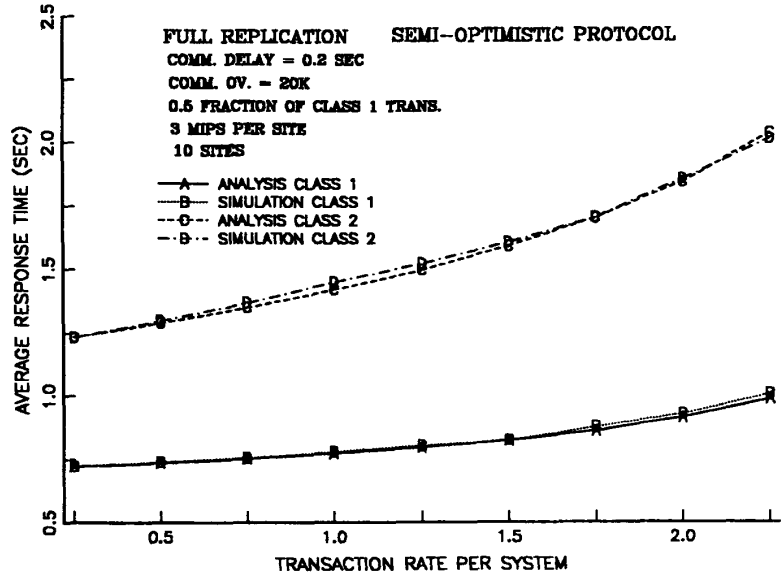

Fig. 1. Full replication simulation and analysis.

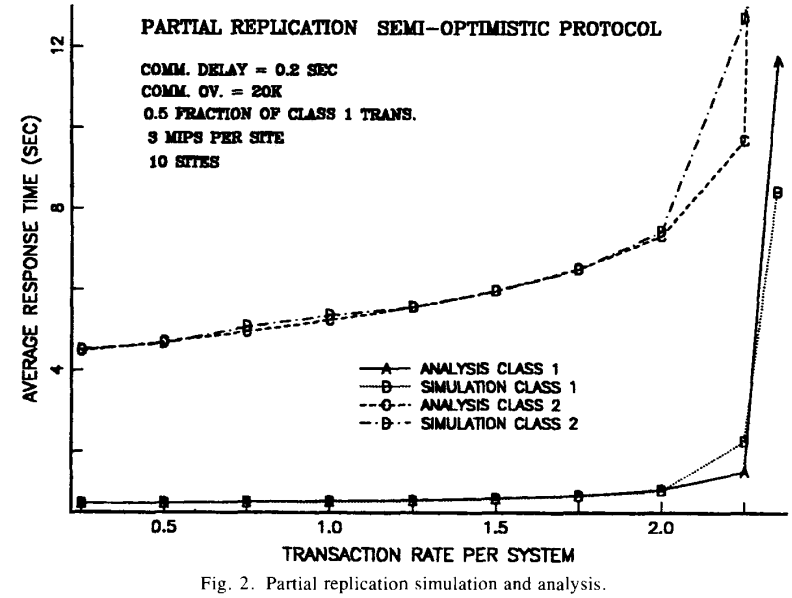

Fig. 2. Partial replication simulation and analysis.

are for the number of replications of 19 and 0, respectively. The negligible communications delay is reasonable for a locally distributed system [8]. Notice from Fig. 4 that, for this case, the response time increases with an increase in the degree of replication for all of the protocols. This is consistent with the observation in [5], where negligible communications delay is assumed. The reason for this behavior is that the reduction in overhead for remote access during the transaction due to the availability of a local copy is overshadowed by the overhead for updating the replicates at commit time. Comparing the response times of the different protocols, we observe that the pessimistic (distributed 2PL) protocol is the best for no replication (for this level of contention), and the pure optimistic protocol has the worst performance. As the number of replications increase, there is a crossover between the 2PL response time and that of the other protocols. This is because the 2PL protocol with primary copy does not benefit much from replication, since communication continues to occur for concurrency control even if

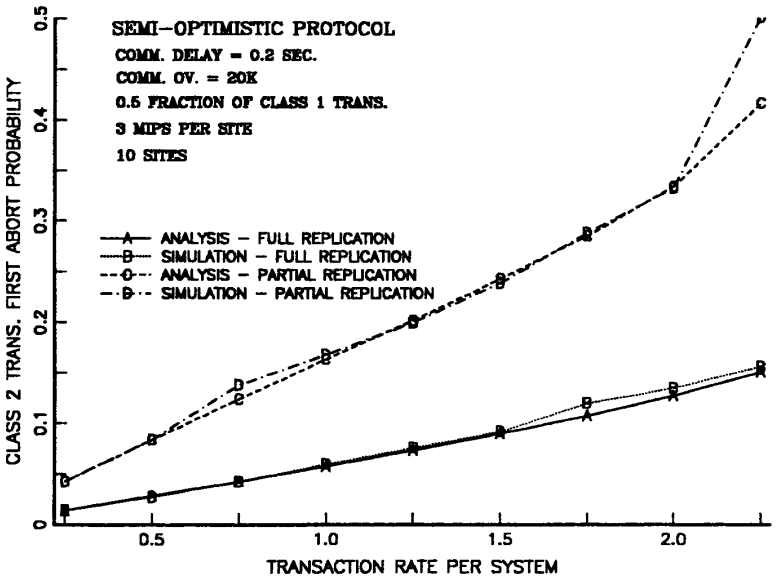

Fig. 3. Abort probability, simulation and analysis.

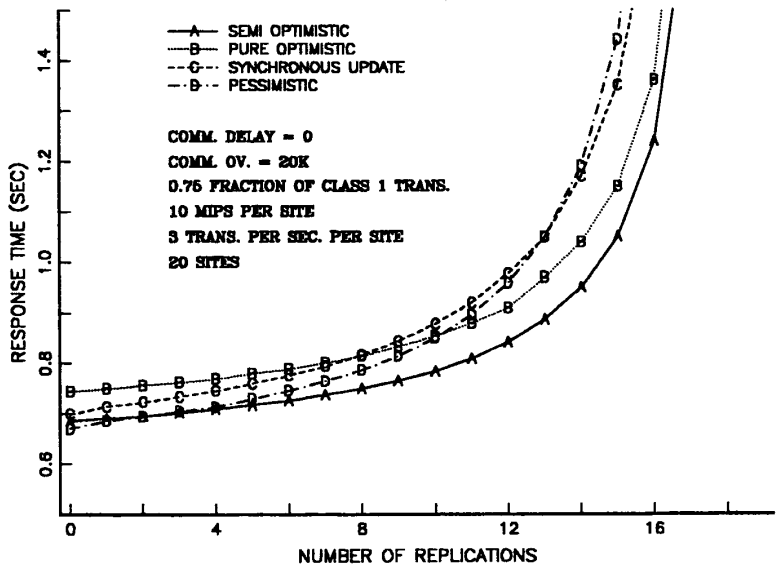

Fig. 4. Response time versus replication, no communications delay.

a local copy is available. All the other protocols do not communicate with the primary copy site during the execution of the transaction except at commit time unless there is no local replication of the data. Therefore, the optimistic type protocols reduce communications during the transaction, and do better than 2PL as the number of replications increases. The semi-optimistic protocol does better than the pure optimistic protocol because it has no aborts for class 1 transactions, leading to smaller CPU utilizations. The performance penalty for synchronous updates for the semi-optimistic protocol can be seen by comparing curves marked *A* and *C* in Fig. 4. This penalty becomes larger as the number of replications increases because the time to update all of the replications increases.

In Fig. 5, we illustrate the effect of a larger communications delay of 0.1 s. For no replication, the 2PL protocol has the smallest response time of the protocols, by a larger margin than that for zero communications delay. As the number of replications increases, the 2PL protocol response time increases as well, for the reason given in

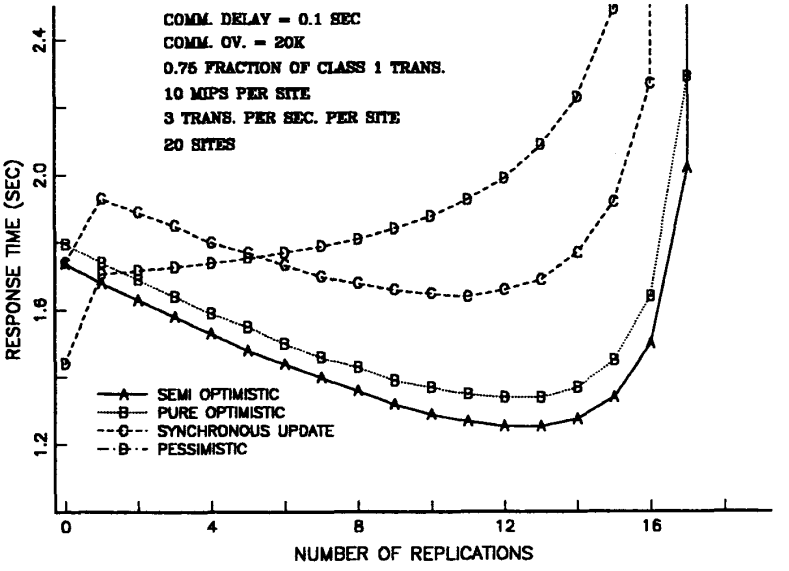

Fig. 5. Response time versus degree of replication, 0.1 s communications delay.

the previous paragraph. However, the response time for the other protocols shows a minimum for partial replication. The reason that a minimum response time is observed for this communications delay, but not for a zero communications delay, is as follows. In the transaction model we use, nonlocal data are accessed by the transaction during its execution, and therefore, serially. Adding replications reduces the overhead for the remote access and eliminates the serial communications delay. On the other hand, at commit time, the replicates are updated in parallel. Therefore, even though the overhead for update propagation is large, the elimination of the serial remote access predominates over the increase in time for parallel updates of the replicates. However, as the number of replicates increases, the overhead for the updates of the replicates continues to grow, and eventually predominates over the reduction in communications delay. Thus, there is an optimal degree of replication. The optimal replication is in general different for the different protocols. Comparing curves marked *A* and *B*, we note that the pure optimistic protocol is a little worse than the semi-optimistic protocol, for the same reasons as in the previous paragraph. Comparing curves *A* and *C*, we observe that the performance penalty for synchronous updates increases with the communications delay. In curve *C*, there is an increase in response time when the degree of replication increases from 0 to 1, before the response time decreases with further increase in the degree of replication. This is because there is a large increase in the update delay (and secondarily, overhead) for even one replicate with synchronous updates. There is smaller incremental update delay with a larger degree of replication.

The effect of larger communications delay is further illustrated in Fig. 6, which is for a 0.2 s communications delay. Comparing this to Fig. 5, we observe that the optimum degree of replication is further accentuated by the increase in communications delay. The optimum degree of replication increases for each of the optimistic type protocols, but this increase in the optimum degree of replication is rather small. The reason is that the optimum is largely determined by the degree of replication at which the CPU's get close to saturation due to the update overhead. With longer communications delay an increase in the degree of replication leads to a larger savings in the serial communications time during the transaction execution relative to the parallel communications for update at commit time, but the overheads for communications are largely the same. Therefore, while the curves in Fig. 6 are sharper, they have a similar optimum degree of replication.

The sensitivity of the optimum degree of replication to the transaction rate per system is illustrated in Fig. 7 for the semi-optimistic protocol. The label of each curve represents the transaction rate at each site. The graph indicates that at low loads, the optimum is at full replication, and as the load increases, the optimum shifts to a smaller degree of replication, and eventually to no replication at all. The reason for this phenomenon is that the update overhead increases with the degree of replication while the overall communications delay decreases. Therefore, if there is spare CPU capacity it pays to replicate, while at high levels of CPU utilization it does not.

The previous observation leads to the question of whether it makes sense to replicate data for any realistic environment with reasonable CPU utilizations. We therefore pose the problem differently. We assume an environ-

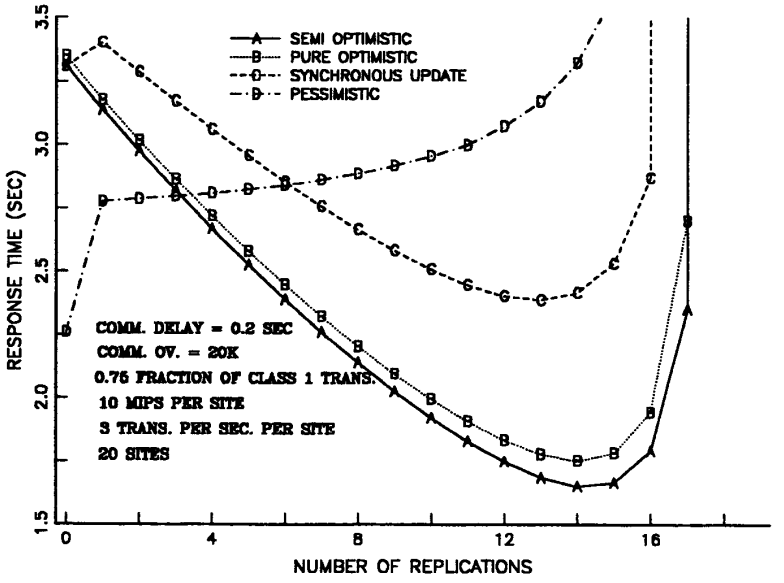

Fig. 6. Response time versus degree of replication, 0.2 s communications delay.

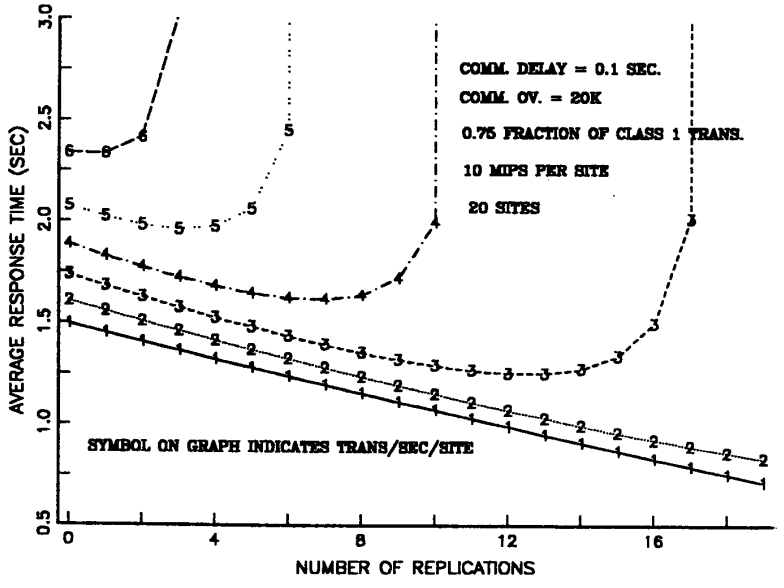

Fig. 7. Sensitivity to transaction rate.

ment with a specific requirement in terms of the transaction rate and a bound on the average transaction response time. We determine the minimum CPU MIPS per site required to meet these requirements of transaction rate and response time. We then examine if there is an optimum degree of replication that results in the lowest total MIPS required to meet these constraints. Posing the problem in this manner has the resultant CPU utilization fall out from the requirements rather than as a stipulation. Fig. 8 shows the minimum MIPS per site required to meet a 2 s transaction response time bound for different degrees of replication, and for the different protocols. For this fairly loose response time requirement, there is no benefit from replication. The reason is that, with no data replication, the loose bound (for this relatively small communications delay) allows a large CPU processing time and consequently results in a small MIPS requirement with a large CPU utilization. Then, the overhead due to updates with

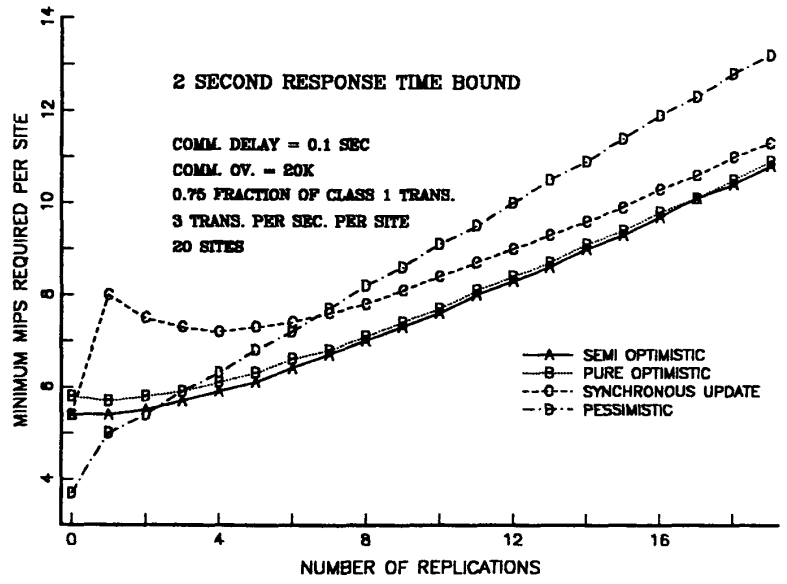

Fig. 8. MIPS required—2 s response time bound.

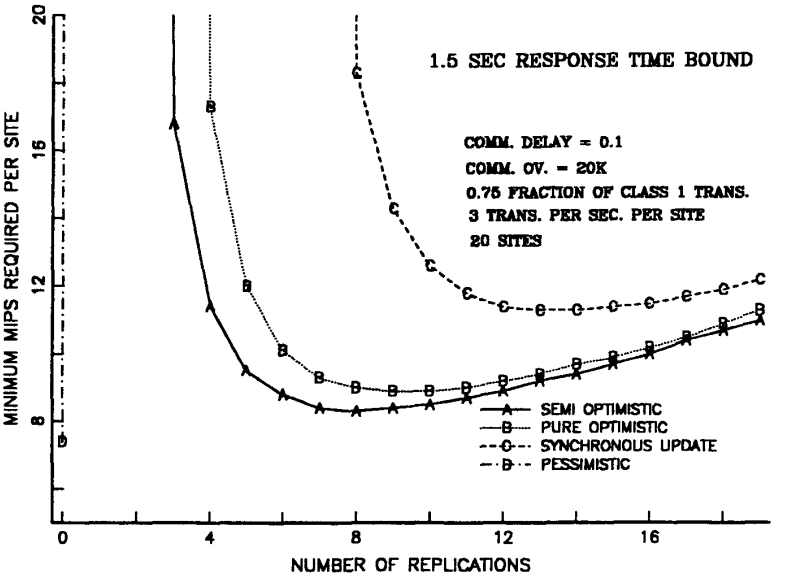

Fig. 9. MIPS required—1.5 s response time bound.

replication requires larger MIPS to prevent CPU saturation. With synchronous updates, the transaction response time increases due to the update time, forcing the CPU to operate at a lower utilization in order to meet the 2 s response time bound; therefore, there is some benefit from replication as the degree of replication increases beyond 1. However, even for this case, the fewest MIPS are required for no data replication. Of all the protocols, the 2PL protocol with no replication requires the fewest MIPS to satisfy the constraints.

The effect of decreasing the response time bound to 1.5 s is shown in Fig. 9, with other parameters remaining unchanged. With this response time bound, the semi-optimistic and pure optimistic protocols are unable to meet the response time constraints without any replication. This leads to an optimum degree of replication. For the 2PL (pessimistic) protocol, the constraints can only be met with no replication. (The near vertical line for this case indicates that for a single replication a very large number of MIPS is required.) Comparing the MIPS required by

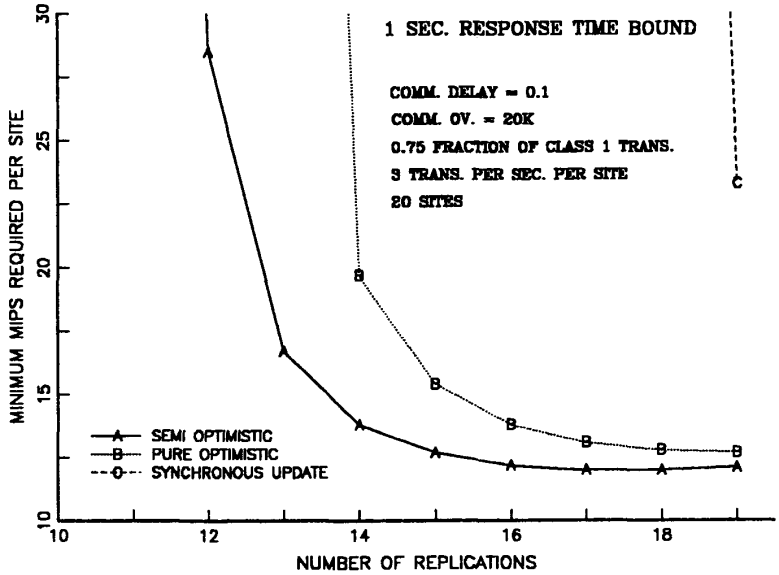

Fig. 10. MIPS required—1 s response time bound.

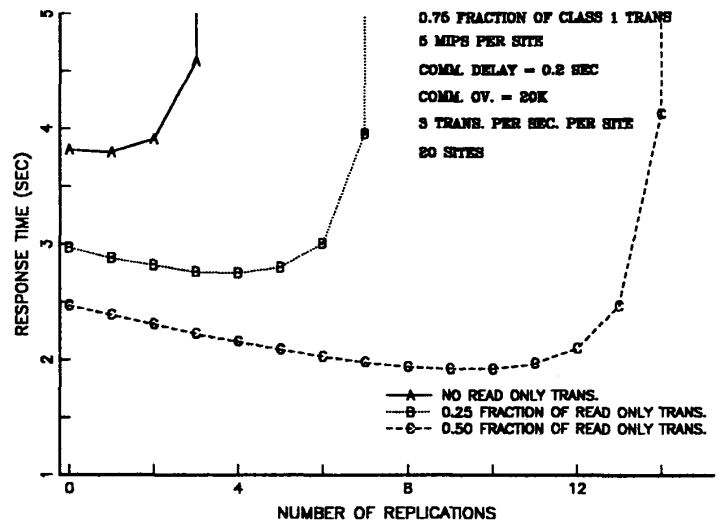

Fig. 11. Sensitivity to fraction of read-only transactions.

the various protocols, the 2PL protocol with no replication wins over the optimistic type protocols at their optimum degree of replication. Decreasing the response time constraint to 1 s leads to the results shown in Fig. 10. The 2PL protocol cannot meet this constraint because of communications delays. For this relatively small response time constraint, full replication leads to the fewest MIPS required for each of the optimistic type protocols. A larger communications delay (not shown) leads to a similar effect as that for decreasing the response time requirement.

The following two charts assume the semi-optimistic protocol with asynchronous updates to do further sensitivity analysis. The effect of transaction mix is next examined. Fig. 11 shows the sensitivity to the number of replications for no read only, 25% read only, and 50% read-only transactions under the semi-optimistic protocol. Transaction rate is assumed to be 3 transactions/s with 5 MIPS at each site. The optimum number of replications increases with the fraction of read-only transactions. Replication becomes increasingly attractive as the fraction of read-only transactions increases. This is because the read-only transactions benefit from local access with replication but do not have the penalty of the overhead for updates.

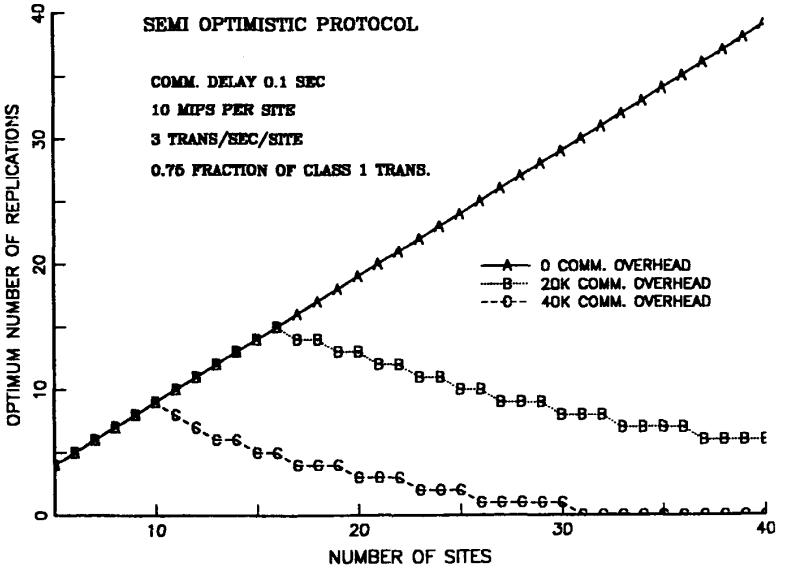

Fig. 12. Sensitivity to number of sites.

Finally, in Fig. 12, we examine the optimum number of replications (in terms of minimum response time) versus the number of distributed sites for different communications overhead. We assume a transaction rate of 3 transactions per second per site and 0.1 s communications delay. With no communications overhead, full replications is the optimum solution. Even in the presence of communications overhead, the optimum can still be full replication if the number of sites is small, but the optimum decreases as the number of sites increases beyond a certain point. This is due to the fact that as the number of sites increases, the average number of sites involved in updates increases for a given number of replications. The larger the communications overhead, the smaller the range where full replication is the optimum.

V. Summary

In this paper, we developed an approximate analytic model to study the tradeoffs of replicating data in a distributed database environment using the primary copy approach for handling data replication. Previous analytic models of replications are for very simple models with read or update requests that do not capture the intricacy of the concurrency control protocol and the effect of communications delay and overhead. The approximate analysis is based on a decomposition approach to examine the effect of hardware resource contention and data contention separately and capture the interaction through an iteration. The accuracy of the approximation is validated through detailed simulations. In addition to the optimistic and pessimistic protocols for concurrency control, we also investigated a semi-optimistic protocol which uses pessimistic protocol for local class 1 transactions and optimistic protocol for the other transactions to reduce the aborts and remote communications. We found that the benefit of replicating data and the optimal number of replicates are sensitive to the concurrency control protocol used. Replications generally degrade the performance under the pessimistic control. Under the optimistic and semi-optimistic protocols, replications can improve response time with an additional MIPS requirement to maintain coherency among the replicates. With replication, the semi-optimistic protocol usually yields the best performance. While synchronous protocols (that update all replicates before committing an update transaction) have worse performance (and better reliability characteristics) than the corresponding asynchronous protocols, they can still benefit from data replication. However, too many replicates usually lead to disastrous performance because of the large overhead for updating all of the replicates. It is thus important to determine the optimal number of replicates. The optimal degree of replication is further affected by the transaction mix (e.g., the fraction of read-only transactions), the communications delay and overhead, the number of distributed sites, and the available MIPS. Various sensitivity analyses have been carried out to examine how the optimal degree of replication changes with respect to these factors. Generally, a low degree of replication is best for cases with loose response time bounds, large communications overheads, and/or low communications delays. The optimal degree of replication increases for small response time bounds, smaller communications overheads, or large communications delays.

References

[1] R. Balter, P. Berard, and P. Decitre, "Why control of concurrency level in distributed systems is more fundamental than deadlock management," in *Proc. 1st ACM SIGACT-SIGOPS Symp. Principles Distributed Comput.*, Aug. 1982.

[2] D. Barbara and H. Garcia-Molina, "How expensive is data replica-

tion? An example," in *Proc. 2nd Distributed Comput. Syst.*, Feb. 1982, pp. 263-268.
[3] P. A. Bernstein and N. Goodman, "Concurrency control in distributed/database systems," *Comput. Surveys*, vol. 13, no. 2, pp. 185-221, June 1981.
[4] ——, "A sophisticate's introduction to distributed database concurrency control," in *Proc. 8th VLDB Conf.*, Sept. 1982, pp. 62-76.
[5] M. J. Carey and M. Livny, "Distributed concurrency control performance: A study of algorithms, distribution, and replication," in *Proc. 14th Int. Conf. Very Large Data Bases*, 1988.
[6] B. Ciciani, D. M. Dias, and P. S. Yu, "On hybrid distributed—Centralized database systems," in *Proc. IFIP Conf. Distributed Processing*, Oct. 1987. New York: Elsevier Science, pp. 523-535.
[7] E. G. Coffman, E. Gelenbe, and B. Plateau, "Optimization of the number of copies in a distributed system," *IEEE Trans. Software Eng.*, vol. SE-7, no. 1, pp. 78-84, Jan. 1981.
[8] D. W. Cornell, D. M. Dias, and P. S. Yu, "On multisystem coupling through function request shipping," *IEEE Trans. Software Eng.*, vol. SE-12, no. 10, pp. 1006-1017, Oct. 1986.
[9] A. Dan, D. D. Towsley, and W. H. Kohler, "Modeling the effects of data and resource contention on the performance of optimistic concurrency control protocols," in *Proc. 4th Int. Conf. Data Eng.*, Los Angeles, CA, Feb. 1988, pp. 418-425.
[10] A. Dan, D. M. Dias, and P. S. Yu, "Database buffer model for the data sharing environment," in *Proc. 6th Int. Conf. Data Eng.*, Los Angeles, CA, Feb. 1990, pp. 538-544.
[11] C. J. Date, *An Introduction to Database Systems, Vol. II*. Reading, MA: Addison-Wesley, 1983.
[12] D. M. Dias, P. S. Yu, and B. T. Bennett, "On centralized versus geographically distributed database systems," in *Proc. 7th Int. Conf. Distributed Comput. Syst.*, Berlin, West Germany, Sept. 1987, pp. 64-71.
[13] B. I. Galler, "Concurrency control performance issues," Ph.D. dissertation, Dep. Comput. Sci., Univ. of Toronto, Sept. 1982.
[13] B. I. Galler and L. Bos, "A model of transaction blocking in databases," *Perform. Eval.*, vol. 3, pp. 95-122, 1983.
[15] H. Garcia-Molina, "Performance of update algorithms for replicated data in a distributed database," Ph.D. dissertation, Dep. Comput. Sci., Stanford Univ., June 1979.
[16] J. Gray, P. Homan, R. Obermarck, and H. Korth, "A straw man analysis of probability of waiting and deadlock," IBM Res. Rep. RJ 3066, San Jose, CA.
[17] IBM Corp., "Customer information control system/virtual storage (CICS/VS): Intercommunication facilities guide," SC33-0133, 1983.
[18] K. B. Irani and H. L. Lin, "Queueing network models for concurrent transaction processing in a database system," in *Proc. ACM-SIGMOD Int. Conf. Management Data*, Boston, MA, Jan. 1979, pp. 134-142.
[19] J. A. Larson and S. Rahimi, *Tutorial: Distributed Database Management*, IEEE Computer Society Press, Silver Spring, MD 20910, 1985.
[20] W. Lin and J. Nolte, "Basic timestamp, multiple version timestamp, and two-phase locking," in *Proc. 9th VLDB Conf.*, Florence, Italy, Nov. 1983.
[21] R. D. Nelson and B. R. Iyer, "Analysis of a replicated data base," *Perform. Eval.*, vol. 5, pp. 133-148, 1985.
[22] D. Potier and P. Leblanc, "Analysis of locking policies in database management systems," *Commun. ACM*, vol. 23, no. 10, pp. 584-593, Oct. 1980.
[23] M. Sinha, P. D. Nanadikar, and S. L. Mehndiratta, "Timestamp based certification schemes for transactions in distributed database systems," in *Proc. ACM SIGMOD Conf.*, Austin, TX, May 1985, pp. 402-411.
[24] M. Stonebraker, "Concurrency control and consistency in multiple copies of data in distributed INGRES," *IEEE Trans. Software Eng.*, vol. SE-5, no. 3, pp. 188-194, May 1979.
[25] U. Sumita and O. R. Liu Sheng, "Analysis of query processing in distributed database systems with fully replicated files: A hierarchical approach," *Perform. Eval.*, vol. 8, pp. 223-238, 1988.
[26] Y. C. Tay, "A mean value performance model for locking in databases," Ph.D. dissertation, Harvard Univ., Cambridge, MA, Feb. 1984.
[27] Y. C. Tay, N. Goodman, and R. Suri, "Locking performance in centralized databases," *ACM Trans. Database Syst.*, vol. 10, no. 4, pp. 415-462, Dec. 1985.
[28] C. Thanos, C. Bertino, and C. Carlesi, "The effects of two-phase locking on the performance of a distributed database management system," *Perform. Eval.*, vol. 8, pp. 129-157, 1988.
[29] R. Williams, D. Daniels, L. Haas, G. Lapis, B. Lindsay, P. Ng, R. Obermark, P. Selinger, A. Walker, P. Wilms, and R. Yost, "R*: An overview of the architecture," in *Improving Database Usability and Responsiveness (Proc. Int. Conf. Databases, Jerusalem, Isreal, June 1982)*, P. Scheuermann, Ed. New York: Academic, 1982, pp. 1-27.
[30] P. S. Yu, D. M. Dias, J. T. Robinson, B. R. Iyer, and D. Cornell, "Modelling of centralized concurrency control in multi-system environment," *Perform. Eval. Rev.*, vol. 13, no. 2 (*Proc. 1985 ACM SIGMETRICS*), pp. 183-191.
[31] P. S. Yu, D. W. Cornell, D. M. Dias, and A. Thomasian, "Performance comparison of IO shipping and database call shipping: Schemes in multisystem partitioned databases," *Perform. Eval.*, vol. 10, pp. 15-33, 1989.
[32] P. S. Yu, D. M. Dias, J. T. Robinson, B. R. Iyer, and D. W. Cornell, "On coupling multi-systems through data sharing," *Proc. IEEE*, vol. 75, no. 5, pp. 573-587, May 1987.
[33] P. S. Yu and D. M. Dias, "Impact of large memory on the performance of optimistic concurrency control schemes," in *Proc. PARBASE-90: Int. Conf. Databases, Parallel Architectures, Appl.*, Miami Beach, FL, Mar. 1990, pp. 86-90.
[34] P. S. Yu and D. M. Dias, "Notes on modelling optimistic concurrency control schemes," IBM Res. Rep. RC 14825, Yorktown Heights, NY, Aug. 1989.

Bruno Ciciani was born in Rome, Italy, in 1955. He received the doctoral degree in electronics engineering from the University of Rome (La Sapienza), Rome, Italy, in 1980.

He is presently a Research Associate with the Dipartimento di Ingegneria Elettronica, University of Rome (Tor Vergata). From 1987 to 1988 he was a Visiting Scientist at the IBM Thomas J. Research Center, Yorktown Heights, NY. His research interests include distributed computer systems, languages for parallel processing, fault-tolerant computing, and performance and reliability evaluation of fault-tolerant systems.

Daniel M. Dias (M'88) received the B.Tech. degree from the Indian Institute of Technology, Bombay, and the M.S. and Ph.D. degrees from Rice University, Houston, TX, all in electrical engineering.

He is a Research Staff Member at the IBM Thomas J. Watson Research Center. His current research interests include database systems, transaction and query processing, parallel and distributed systems, interconnection networks, and performance analysis.

Philip S. Yu (S'76-M'78-SM'87) received the B.S. degree in E.E. from the National Taiwan University, Taipei, Taiwan, Republic of China, in 1972, the M.S. and Ph.D. degrees in E.E. from Stanford University, Stanford, CA, in 1976 and 1978, respectively, and the M.B.A. degree from New York University, New York, NY, in 1982.

Since 1978 he has been with the IBM Thomas J. Watson Research Center, Yorktown Heights, NY. Currently he is manager of the Architecture Analysis and Design group. His current research interests include database management systems, parallel and distributed processing, computer architecture, performance modeling, workload analysis, and computer networking. He has published about 70 papers in refereed journals and conferences.

Dr. Yu is a member of the Association for Computing Machinery.