A Hybrid Distributed Centralized System Structure for Transaction Processing

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

Abstract—In a fully centralized transaction processing system, all transaction input messages are shipped to the central system, where the transaction is processed, and output messages are sent back to the terminal; hence, the centralized system has this communications overhead and delay regardless of geographical locality of reference. On the other hand, the performance of a fully distributed system depends critically on the number of remote data accesses by a transaction; the performance of the distributed system is better than the centralized system if the number of remote data accesses per transaction is small, but is much worse otherwise. In this paper a hybrid system structure is examined that consists of distributed systems to take advantage of locality of reference, and a central system to handle transactions that access nonlocal data. We note that several transaction processing applications such as reservation systems, insurance, and banking have such regional locality of reference. A concurrency and coherency control protocol is described that maintains the integrity of the data and has good performance for transactions that access local or nonlocal data. It is shown that the performance of the hybrid system is much less sensitive to the fraction of remote accesses than the distributed system, and offers similar performance to the distributed system for local transactions.

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

I. Introduction

TRADITIONALLY, transaction processing systems have consisted of a large central computer complex connected to the user terminals by a computer network. In recent years there has been considerable interest in a geographically distributed database approach, in which the databases are partitioned and distributed among regional processing systems, and some "request shipping" mechanism is provided to support the access of data in a nonlocal database partition [5], [13], [15], [22], [24]. There are a number of tradeoffs between the two approaches [9]. In this paper we examine a hybrid system, consisting of both distributed systems and a central complex, and we attempt to provide the best features of both approaches for environments where there is significant locality of reference. Several transaction processing applications such as reservation systems, banking, and insurance exhibit such locality of reference.

A number of factors have motivated interest in the distribution of the data and computing power [12]. As the size of the database and the volume of processing increases, the capacity of a single system may be exceeded, and thus the processing needs to be distributed between two or more systems. These systems may be collocated in the same machine room, and linked by a high speed network, thus extending the centralized approach. Alternatively, the system may be geographically distributed and interconnected by a long haul network. In the centralized approach, at least one input message must be sent to the central complex across the network, followed by processing at the central site, and return messages across the network. In the distributed system, the input and output messages can be local to the distributed system, allowing a potential reduction in communications delay and cost. However, data required by a transaction may not be available locally, and it may be necessary to obtain the required data from a remote system. Further, if the database needs to be updated, then every system involved in an update must participate in transaction commit processing [7], which also requires the exchanging of messages between systems. These communications during the processing of the transaction for the distributed system may overshadow the communications needed to communicate input and output data to the user in the centralized systems. Thus, the tradeoff between the approaches in terms of communications cost depends critically on whether transactions, originating at a computer of the distributed system, largely refer to and update data available locally. In [9] it was shown that there is a cross-over in the response time of the two approaches as the number of remote calls per transaction increases. The distributed system may offer a small improvement in response time if the number of remote calls per transaction is small, but will have a much poorer response time in many other cases. The cross-over point depends on several factors, including the communications delay and overhead, data contention level, and the average number of systems in commit processing. At high contention levels and typical communications delays, the number of remote calls must be very small for the distributed approach. Hence, the application must be very stable in this respect, since a small

Manuscript received July 7, 1986; revised March 1, 1990.

B. Ciciani was with the IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598. He is now with the Departimento Di Ingegneria Elettronica, Seconda Universite Di Roma, Rome, Italy.

D. M. Dias and P. S. Yu are with the IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598.

B. R. Iyer is with the Database Technology Institute, IBM Corp., San Jose, CA 95161.

IEEE Log Number 9036261.

increase in the remote calls could make the system unstable.

Given these tradeoffs between the distributed and centralized approaches, we consider the hybrid system that has distributed computer systems to take advantage of locality of reference, and a central system to handle transactions that access global data. In Section II we describe the hybrid system protocols, and qualitatively examine their potential. Section III describes a model for comparing the approaches. Results and further discussion appear in Section IV, and concluding remarks in Section V.

II. Hybrid System Protocols and Qualitative Comparison

The scenario that we examine is illustrated in Fig. 1. We assume that user work-stations or terminals are connected to geographically distributed computer systems. This may be a local or metropolitan area network. The distributed systems in turn are connected to a central computing complex through a long-haul communications network. We compare this to a purely distributed system, without the central complex, and to a purely centralized system, in which the distributed computer systems are replaced by communications concentrators. In this section we first describe a control protocol for the hybrid system, and then qualitatively compare the approaches.

A. Control Protocol

The control protocol of the hybrid system is critical to its performance. Without a good protocol, the performance can be worse than either a purely distributed or a purely centralized system. We distinguish two types of transactions. Class A transactions are purely local, and do not need any data from any other site. The second class B of transactions need to access nonlocal data. We assume that with some preprocessing it is possible to identify the class of a transaction. To preserve the advantage of distributed systems, we would like to run class A transactions on the local system. In order to improve the stability of the system, we assume that class B transactions (rather than database calls) are shipped to the central system in their entirety. Thus, under the hybrid system protocol, a class A transaction will be referred to as a local transaction and a class B transaction as central transaction. (Our analysis is restricted to single input/single output class of transactions.)

First, we observe that a class B transaction running at the central site should be able to principally access data local to the central site, otherwise its performance would be worse than that of the distributed system. This means that (at least some) data at the local systems must be replicated at the central system. Second, it should be possible to commit class A (purely local) transactions without waiting for communications with the central site; otherwise, the commit communications delay would make it preferable to ship the entire transaction to the central site, eliminating the advantage of the distributed system. This means that when data that is replicated at the central and

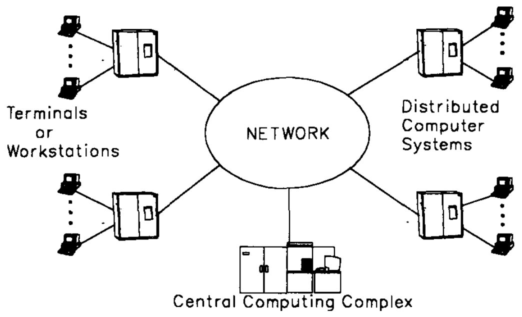


Fig. 1. Hybrid system structure.

distributed site is updated at the distributed site, the update must be propagated asynchronously to the central site. That is, a class A transaction at a local site commits without waiting for a communication to the central site, and the update at the central site is done later. Consequently, the data at the central and distributed sites are not always coherent, and the control protocol must ensure that transactions use consistent data. Since the central system obtains updates from local systems asynchronously, a transaction running at the central system cannot commit until the protocol ensures that there is no concurrent update at a distributed system.

The protocol is as follows. As stated above, the database is partitioned among the distributed systems, and is replicated at the central site; each distributed system is concurrency control master for its database partition. Class A transactions (identified after preprocessing) access data entirely from the system's database partition, and make local lock requests for data required (shared or exclusive mode). The lock manager maintains two fields for each lock—a concurrency control field (shared or exclusive) and a coherency control field (used to maintain coherence of data between distributed and central sites). The concurrency control field is used with usual locking protocols to ensure that compatible lock requests are granted and that incompatible lock requests are queued. The coherence control field is used to maintain a count of outstanding responses (see below) and is initially set to zero. At transaction commit point, the share or exclusive lock indicated in the concurrency control field is released by the transaction (transactions queued on the entity can then be granted the lock); further, the coherence field count of the lock is incremented for updates to indicate a pending response from the central site. Asynchronously, a message is sent to the central site indicating the locks released by the transaction, and a copy of the blocks updated by the transaction is sent along with this message. When the central site responds that it has processed the message (as detailed below) the coherence field pending message count is decremented. Meanwhile the local transaction completes without waiting for this message to be sent or response received, and thus provides good response time for purely local transactions. Further, another local transaction can lock and update the data, increment the coherence count and send another asynchronous update message to the central site before the acknowledgment from the first message is received. These asynchronous messages may also be batched to reduce the overheads involved. Note however, that the communications protocol must ensure that these asynchronous messages are delivered and processed at the central site in the order that they were originated. If the communication mechanism itself does not ensure this, then a higher level protocol must ensure that if a transaction commits and has locks on items with nonzero coherence count, then (although the transaction can commit) the asynchronous messages to the central site are queued until the previous messages to the central site are acknowledged.

Class B transactions are shipped to the central site where they access (replicated) data locally. At the central site, local locks are used for concurrency control among transactions running at the central site. However, as described above, transactions at the distributed site may concurrently use and update data, that is propagated asynchronously to the central site. When the central site receives a message from the distributed sites indicating locks obtained and updates, any locks at the central site on the data are invalidated (transactions holding these locks are marked for abort), the data is updated and an acknowledge is sent back to the distributed site. At transaction commit time, the central site checks if the transaction has been marked for abort due to invalidated locks held; if not, the central site simultaneously sends, to all the sites that are masters of the data locked, a list of the locks and mode required along with a copy of the block updated by the transaction. The distributed sites examine the lock status (both concurrency and coherency); if the locks requested are compatible and if the coherency control status is null (i.e., no in-flight asynchronous updates to the data) then the locks are granted, otherwise a negative acknowledge is sent to the central site. If the central site acquires all locks at the distributed sites for the transaction, it checks to ensure that the acquired locks were not invalidated by asynchronous updates at distributed sites, and then it sends a commit message to all the involved sites; if not, it reexecutes the transaction and repeats the process. When the distributed sites receive the commit message, the database is updated and the locks corresponding to the transaction at the central site are released.

The above protocol is readily understood by noting that concurrency control among transactions at the same site is done using locking, while an optimistic concurrency control is used for managing concurrency control between transactions running at the local and central sites. We note that the advantage of optimistic type concurrency control for intersite concurrency control is also discussed in [2]. In our protocol, conflicts among transactions running at the same site result in a wait until the lock on the conflicting data item is released. Conflicts between transactions running at the local and central sites results in aborts of the centrally running transaction. The details of the protocol as stated above are principally to ensure that conflicts between local and centrally running transactions are correctly detected, and the conflicting central transaction aborted. Essentially, at the end of its execution, a central transaction may abort either due to an asynchronous message received from a local site that indicates a conflicting access by a committed local transaction, or at commit time when it communicates with the local sites that are masters of the data that it accessed and determines that a conflict occurred. In the latter case there are two possibilities that cause aborts: 1) a locally running transaction is holding a conflicting lock on the data, or 2) a local transaction has committed and is in the process of asynchronously informing the central site of the update. The former possibility is easily determined, while the latter possibility is determined by keeping track of a pending update to the central site, as detailed above.

In the above we assumed that an incoming transaction is preprocessed to determine whether it is of class A or B. In some applications this determination is straightforward. For instance, in a banking application, the user account number can be used for this determination, or in an airline reservation system, the flight number, or the policy number for an insurance application. The hybrid system is particularly suited to applications with significant regional locality, as in the above cases, where the determination of the class of the transaction is relatively easy. Such preprocessing with some simple estimation can also be used for load sharing, in that class A transactions could be shipped to the central site in case the local site is overloaded due to system dynamics [3]. The protocol as stated above stipulates that all aborts of central transactions occur only at commit point, i.e., transactions are not aborted as soon as possible. In [26] it is shown that with reasonably large database buffers so that data read in from secondary store during the first run of a transaction is available in the buffer during the rerun of a transaction, aborting a transaction at commit point rather than immediately is preferable. Therefore, the protocol is stated and analyzed for aborts at commit point, although the specification and analysis is easily modified for immediate aborts of transactions [26].

B. Qualitative Comparison

There are a number of tradeoffs between the purely centralized, the fully distributed and the hybrid approaches. For class A transactions, the processing will be done at the local site in the cases of the hybrid system and the distributed system, thus incurring no communications delay whereas in the centralized system, a round trip communications delay is incurred for shipping the transaction to the central site for processing. While class A transactions for the hybrid system do not wait for communications to the central site, updates by these transactions are asynchronously propagated to the central site, incurring communications and processing overheads. Furthermore, in the hybrid system coherency could cause aborts of class B transactions at the central site.

Next consider Class B transactions. There is no distinction between class B and class A transactions in a centralized system. In the fully distributed system, a remote database call has to be issued to access a nonlocal database for class B transactions. One issue is that of concurrency control. Each system owns a partition of the database and provides concurrency control for the partition it owns. It obtains locks for all data in its partition; thus granting of locks is localized to this system. However locks are held during all communications for remote data access, update, and commit processing, whereas in the centralized system no locks are held while the user input or output data is communicated to the central complex. This gives rise to larger data contention for the distributed system, particularly important at high contention levels. In the hybrid system, the class B transaction is shipped to the central site. Since there are two copies of the database, an optimistic approach is adopted for concurrency control at the central site to reduce frequencies of communications while confirmation on locks is done at commit time. Compared to the distributed system, the communications delay during a transaction for remote data access and update is eliminated but the commit process may be more complex and takes somewhat longer. The saving in lock holding time can be substantial as the number of remote calls per transaction in the distributed system increases. However, in the hybrid environment class B transactions can get aborted and require reexecution.

Another issue is the small system effect [8]. As the processing is spread across multiple sites, each computer system may be relatively small. This would mean that a transaction would take longer to run because of the slower processors; further, since resources would be held for a longer time, the data contention would be larger. This is in contrast to centralized system which has the opportunity of employing large processors. Hybrid systems can at least use a larger processor at the central site than fully distributed systems. Furthermore, load balancing is easier to manage in the centralized system or the central system in the hybrid system than for the distributed system or the local system in the hybrid system.

An important performance issue is that of the sensitivity of system performance to workload parameters, since there could be changes in the workload for the same application. Also the systems may need to support applications with significantly different characteristics in the future. The distributed system is particularly sensitive to the fraction of remote cells, whereas the hybrid system alleviates this problem, as we will see in the next section.

III. Methodology

The methodology described below is used to evaluate the transaction response times and processor utilizations for distributed, centralized and hybrid systems. For an equitable comparison, each system is subject to the same total transaction rate, and required to meet the same (parameterized) response time bound. An approximate analytical model is developed to evaluate the transaction response time based on the MIPS and transaction characteristics derived from real workloads and assumed locality among transactions arriving on different systems.

The methodology is an extension of earlier work for centralized database systems using locking reported in [23], [25] and distributed databases without replication also using locking [9]. 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 [21] for locking in centralized databases, using the steady state average values of variables rather than probability distributions of the variables. As in [25] we decompose the resource and data contention and capture their interaction using an iteration. A similar decomposition approach for optimistic concurrency control in centralized systems has been reported in [26]. (A slightly different approach using a piecewise linear model to capture resource contention is used in [6].) In [27], 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. In most of the previous work, the interaction between data and resource contention is either ignored or the resource contention is not modeled, [14], [19], [11], [10], [21]. Previous models of distributed databases using replication have either used a very simple read write model and ignored resource contention [4], [18] or have resorted to simulation [1], [2].

A. Trace Analysis

Many high volume database systems currently run on single systems. Can such high volume operations be sustained by using less total MIPS on a distributed or a hybrid system? To answer this question, traces of transaction activity during periods of peak utilization were taken from two large single system installations running IBM's IMS database system [20]. The first was an inventory parts database traced for 15 minutes, and the second was an online materials planning database system traced for 60 minutes. Both traces lead to similar results. The results presented in this paper are for the first trace.

The probability of lock contention was found to be about 0.0038 per lock request by using the derived trace to drive a simulation of two coupled 14 MIPS systems, each subject to an average workload of 20 transactions per second. This lock contention probability is projected to other system configurations by the approximate analytical model presented below.

B. The Model

The approximate model used for a fully distributed and a purely centralized transaction processing system is the same model as that described in [9]. The model for the hybrid system is described below. At the end of the section, the differences between this model and the models for the fully distributed and centralized systems are given. As outlined in the scheme above, transactions at the local site use locking and could contend with other transactions running locally and with transactions running at the central site that are in their commit phase. First, we look at the evaluation of the contention and abort probabilities of transactions running at the local site and central sites. A glossary of the notation used appears in Table I.

As described in a previous section, transactions that run on a local site use locking for concurrency control. Transactions assigned to the central site use locking for concurrency control to resolve conflicts with other transactions running at the central site, and an optimistic method to resolve conflicts with transactions running on the local site. In essence, transactions running at the central site are aborted if they conflict with transactions running at any local site (we will see later that this is not strictly true). In this section, we develop the equations used to determine the probability of contention and abort. For ease of presentation, we make more assumptions than needed, and relax these assumptions in a sequence of steps.

Because of the difference in the analysis of their response times, we distinguish three kinds of transactions: 1) transactions assigned to run at local sites, with expected response time denoted as R_L, 2) newly arrived transactions assigned to the central site (that run for the first time), with response time denoted as R_{C1}, and 3) transactions that run at the central site after being aborted at least once (rerun transactions) with expected response time denoted as R_{C2}. R_L, R_{C1}, and R_{C2} denote only the portion of the transaction response time during which transactions could hold locks. All locks are assumed to be held until the end of the transaction. For the estimation of contention and abort probabilities, we are leaving out a fixed initial portion of the transaction response time, during which the transaction goes through a transaction set-up phase (during which no locks are held). A transaction that is rerun at the central site after an abort is modeled to find all data referenced in its main memory. The difference between R_{C1} and R_{C2} is the I/O time. Another assumption is that locks are acquired uniformly during the portion of the response time defined above. Furthermore, all lock requests are assumed to be exclusive. Generalization to multiple lock modes like shared and exclusive is straightforward [21], [28].

First we assume that communications between central and local systems and authentication at the local sites are instantaneous for the central transactions. It is possible to view the hybrid system concurrency control algorithm as one of pure locking with the modification that lock acquisitions by the local and central sites are not communicated to each other until the transaction that acquired the lock is preparing to commit. Let us consider any particular order of transactions running in a hybrid system. We want to estimate the total number of conflicts among transactions. If there were an oracle that observed the lock request arrivals at each local site and central site and communicated them to each other, then every conflict could be translated to a lock contention. (This would be the pure locking scheme.) Among two conflicting transactions, one

TABLE I
GLOSSARY OF NOTATION

λ	Transaction arrival rate at each distributed site
N_{DS}	Number of distributed systems
p_{LOC}	Fraction of class A (local) transactions
$P_{C,CEN}$	Conflict probability for a central transaction
$P_{CEN,CEN}$	Prob. of central trans. contention with another central transaction
$P_{CEN,LOC}$	Prob. of central trans. contention with a local transaction
$P_{CEN,CEN1}$	Prob. of central trans. contention with new central trans.
$P_{CEN,CEN2}$	Prob. of central trans. contention with rerun central trans.
$P_{C,LOC}$	Conflict probability for a local transaction
$P_{LOC,CEN}$	Prob. of local trans. contention with a central transaction
$P_{LOC,LOC}$	Prob. of local trans. contention with a local transaction
$P_{LOC,CEN1}$	Prob. of local trans. contention with new central trans.
$P_{LOC,CEN2}$	Prob. of local trans. contention with rerun central trans.
P_l	Local trans. lock contention prob. under hybrid system protocol
P_c	Central trans. lock contention prob. under hybrid system protocol
p_A	Probability of first abort of central transaction
$\tilde{p}_A$	Probability of abort of a rerun central transaction
$P^{COMIT}_{LOC,CEN}$	Prob. of contention of a local trans. with a central trans. in its authentication phase
P_{GI1}	Prob. that first abort is detected before comm. to local sites
P_{GI2}	Prob. that subsequent abort is detected before comm. to local sites
R_{TL}	Average total local transaction response time
R_L	Average local transaction response time while holding locks
$R_{CPU,L}$	Average CPU service and queueing time in R_{TL}.
$R^{INIT}_{CPU,L}$	Average initial processing time in $R_{CPU,L}$.
$R^{PROC}_{CPU,L}$	Average transaction application processing time in $R_{CPU,L}$.
$R_{IO,L}$	Average I/O time in R_{TL}.
t_{IO}	Average time per I/O
n_{IO}	Average number of I/Os per transaction
n_{IOPI}	Average initial I/Os per transaction
n_{IODB}	Average database I/Os per transaction
R_{IOPI}	Average time for n_{IOPI} I/Os
R_{IODB}	Average time for n_{IODB} I/Os
$R_{CONT,L}$	Average time in lock contention wait in R_{TL}.
R_{TC}	Average central (shipped) transaction response time
R^{SHIP}_{CPU}	Average CPU component in R_{TC} for determining transaction type and shipping transaction and result
R_{COMDLY}	Average communication delay in shipping trans. to central site and getting the results back
$R^{INIT}_{CPU,C}$	Average initial processing time in R_{TC}.
R_{C1}	Average first run time for central transaction while holding locks
R_{C2}	Average second run time for a central transaction
R_{COM}	Av. Comm. delay between central and local sites
$R^{PROC}_{CPU,C}$	Average transaction application processing time in R_{C1}
$R_{CONT,C}$	Average time in lock contention wait for central trans.
N_l	Number of locks per transaction
R_{HOLD}	Average time lock held at local site by central trans. in its authentication phase
β_1	Avg. period during which locks are acquired for 1st run central trans.
γ_1	Average period during which no further central locks are acquired for 1st run central trans.
β_2	Avg. period during which locks are acquired for 2nd run central trans.
γ_2	Average period during which no further central locks are acquired for 2nd run central trans.

TABLE II
CONFLICT RESOLUTION UNDER LOCKING

Lock Holder / Lock Requestor	Central Tx.	Local Tx.
Central Tx.	Central Req. Suspend.	Central Req. Suspend.
Local Tx.	Local Req. Suspend.	Local Req. Suspend.

would suspend and wait for the lock to be released, as summarized in Table II. Let us first study the conflict probability in this environment with the oracle, and then map it back to the original environment. If the combined lock request pattern as seen by the oracle were to be submitted to a single lock manager then the probability of contention per lock request could be projected as below. We start from the lock contention probability $\overline{P}_c$, that is observed in the trace driven simulation of a centralized system, subject to a transaction rate of $\overline{\lambda}$. The expected response time, when holding locks, for transactions in the measured system is denoted as $\overline{R}$. Based on the asymptotic results of the analysis of Mitra and Weinberger [17] and Lavenberg [16] and the general results in [28], we project lock contention to be directly proportional to 1) transaction rate per database, 2) number of locks/transaction to a database (N_L) and 3) expected holding time for a lock as in [23], [25]. That is to say $\overline{P}_C = C(\overline{\lambda} N_L \overline{R}/2)$, for some constant C. The term $\overline{R}/2$ is the expected lock holding time assuming that a transaction acquires locks uniformly over the time that it holds locks.

We define lock contention probability for a centrally running transaction $P_{C.CEN} = P_{CEN.CEN} + P_{CEN.LOC}$, where $P_{CEN.CEN}$ is the probability that the transaction with which the conflict occurs (the current lock holder) is a centrally running transaction, while $P_{CEN.LOC}$ is the probability of a similar conflict with a local transaction holding the lock desired. Similarly $P_{C.LOC}$ is defined as the lock contention per lock request for a local transaction. $P_{C.LOC} = P_{LOC.CEN} + P_{LOC.LOC}$, where $P_{LOC.CEN}$ is the probability of contention with a central transaction and $P_{LOC.LOC}$ is the probability of contention with another local transaction. We divide $P_{LOC.CEN}$ into $P_{LOC.CEN1}$ and $P_{LOC.CEN2}$, as contention probabilities due to whether the central transaction, which held the lock and caused the contention, being a new transaction or a rerun transaction, respectively.

Let N_{DS} be the number of local systems. Each local system has a local database and is subject to a transaction rate λ. Of these transactions, a fraction p_{LOC} executes locally, thus offering a transaction arrival load of $\lambda(1 - p_{LOC})N_{DS}$ at the central site. The central site has a copy of each database in the distributed systems. The transactions arriving at the central site are assumed to access the databases at the central site uniformly. Based on our assumptions about the growth in lock contention probability, we may write:

$$\begin{aligned} P_{CEN.CEN} &= P_{CEN.CEN1} + P_{CEN.CEN2} \\ P_{CEN.CEN1} &= C\left\{\lambda N_{DS}(1 - p_{LOC})N_L \frac{R_{C1}}{2}\right\} \\ P_{CEN.CEN2} &= C\left\{\lambda N_{DS}(1 - p_{LOC}) \frac{p_A}{1 - \tilde{p}_A} N_L \frac{R_{C2}}{2}\right\}. \end{aligned} \tag{1}$$

Equation (1) is the sum of two product terms. In the first term, $P_{CEN.CEN1}$, the probability of contention with a new central transaction, $\lambda N_{DS}(1 - p_{LOC})$ is the new transaction arrival rate at the central site. Each transaction makes N_L lock requests per transaction. The execution time of a new transaction is R_{C1}. The second term, $P_{CEN.CEN2}$ is the probability of contention with rerun transactions at the central site. We define p_A as the probability that a central transaction aborts at the end of its first execution, and $\tilde{p}_A$ as the abort probability for any rerun transaction. Central transactions reexecute at the rate $\lambda N_{DS}(1 - p_{LOC})p_A/(1 - \tilde{p}_A)$.

Similarly, we may write down the expression for central-local, local-central, and local-local conflicts as,

$$P_{CEN.LOC} = C\lambda N_{DS} p_{LOC} N_L \frac{R_L}{2} \tag{2}$$

$$P_{LOC.CEN1} = C\lambda N_{DS}(1 - p_{LOC})N_L \frac{R_{C1}}{2} \tag{3}$$

$$P_{LOC.CEN2} = C\lambda N_{DS}(1 - p_{LOC}) \frac{p_A}{1 - \tilde{p}_A} N_L \frac{R_{C2}}{2} \tag{4}$$

$$P_{LOC.LOC} = C\lambda N_{DS} p_{LOC} N_L \frac{R_L}{2}. \tag{5}$$

We now consider the hybrid system concurrency control protocol (without the help of an oracle that has global information). Again we assume that the communications time between central and local systems and authentication at the local sites are instantaneous for the central transactions. As has been argued before, under the hybrid system protocol, a locally running transaction only contends with other locally running transactions. Centrally running transactions contend with other central transactions, and the conflicts between local and central transactions translate into aborts of the central transaction, as summarized in Table III. In the hybrid system, we may write the local transaction lock contention probability, $P_L = P_{LOC.LOC}$, and the central transaction lock contention probability, $P_C = P_{CEN.CEN}$.

Under the hybrid system protocol, the terms central-local conflict and local-central conflict have slightly different meaning now without the help of an oracle. Conflicts between central and local transactions are only detected at the end of the central transaction execution. Central transactions can get aborted for two types of situations. One is that the locks requested by the authenticating central transaction is currently held by some local transaction. This is referred to as the central-local conflict. The other is that the data items accessed by the central transactions have become obsolete. That is to say, they have been updated by some other local transaction. $P_{LOC.CEN1}$ (respectively, $P_{LOC.CEN2}$) now represents the probability that an update request from a committing local transaction can conflict with a new (respectively, rerun) central transaction. This is referred to as a local-central conflict since it has the same expression as before. (Note that the property that the transaction conflict expression is the same under either the locking or the optimistic concurrency control schemes is referred to as the *conservation* property which is discussed in detail in [28].) The rate of local-central conflicts between local transactions and a new central transaction is $p_{LOC.CEN1}N_L\lambda N_{DS}p_{LOC}$. These conflicts result in aborts, and are distributed over $\lambda N_{DS}(1 - p_{LOC})$ new central transactions per second. Hence

$$p_A = p_{LOC.CEN1}N_L \frac{p_{LOC}}{(1 - p_{LOC})} + P_{CEN.LOC}N_L \tag{6}$$

where the second term is due to aborts caused by the authenticating central transaction requesting a lock, already held by a local transaction in an incompatible mode. Similarly, we can show that

$$\tilde{p}_A = P_{LOC.CEN2}N_L \frac{p_{LOC}(1 - \tilde{p}_A)}{(1 - p_{LOC})p_A} + P_{CEN.LOC}N_L. \tag{7}$$

We now relax the assumption that authentication of central transactions is instantaneous. Assume that there is a finite communications delay R_{COM} between the central and local sites. First, note that no further locks are ac-

TABLE III
CONFLICT RESOLUTION UNDER HYBRID SYSTEM PROTOCOL

Lock Holder / Lock Requestor	Central Tx.	Local Tx.
Central Tx.	Central Req. Suspend.	Central Req. Abort.
Local Tx.	Central Req. Abort.	Local Req. Suspend.

quired at the central site during this authentication phase. Thus R_{C1} consists of a time β_1 (the execution phase of the transaction) when locks at central sites are acquired, and a time γ_1 for authentication during which no further locks at central sites are acquired. Similarly, β_2 and γ_2 can be defined for R_{C2}. Next, let us assume that during authentication the central transaction will hold locks at the distributed site for a time period R_{HOLD}. (R_{HOLD} includes a communications delay of $2R_{COM}$, i.e., to and from the central site, and CPU time at central and local sites for communications and commit overheads.) Note that, of the centrally running transactions that abort, some abort because of being informed from one of the local sites about the need to abort, and the others abort after they go into authentication phase and find out from one of the local systems that they have suffered a conflict. Let us assume, that for central transactions executing for the first time, the fraction of the first kind of abort is P_{GL1} and the second kind is $1 - P_{GL1}$. For rerun transactions at the central site, we will name these quantities P_{GL2} and $1 - P_{GL2}$, respectively. It will be shown later how to estimate P_{GL1} and P_{GL2}. Note that, of the aborting transactions, there is only a fraction that go to the local site for authentication. Thus γ_1 can be estimated as $(1 - p_{GL1}p_A)R_{HOLD}$. Now, we estimate the contention experienced by local transactions as

$$P_L = P_{LOC.LOC} + P_{LOC.CEN}^{COHER},$$

$$p_{LOC.CEN}^{COHER} = CN_L\lambda N_{DS}(1 - p_{LOC})\left\{1 + (1 - p_{GL1})p_A + \frac{p_A\tilde{p}_A}{1 - \tilde{p}_A}(1 - p_{GL2})\right\}R_{HOLD}.$$

We are assuming that all central transactions going into authentication at all local sites will hold locks. Actually, if a conflict is detected at any local site, then the locks will not be held for the remaining of the authentication process. Hence, our estimate for local site lock contention is a slight over estimation.

We now consider the effect of R_{COM} on computing the contention probabilities for $P_{LOC.CEN1}$ in (3). The communications delay has the effect of increasing by $2R_{COM}$ the interval during which local transaction updates cause the data accessed by a central transaction to become obsolete. $R_{C1}/2$ is replaced by $\beta_1/2 + 2R_{COM}$. Similar changes apply to R_{C2} and the computation of $P_{LOC.CEN2}$ in (4). For $P_{CEN.CEN1}$, the central lock held time, $R_{C1}/2$, in (1) is replaced by $\beta_1/2 + \gamma_1$. Similar changes applies to $P_{CEN.CEN2}$.

In order to estimate P_{GL1} and P_{GL2}, let us again first ignore the communications delay. Consider a particular conflict between a central and a local transaction. Under the hybrid system concurrency control protocol, we become aware of conflicts only when the first of the two conflicting transactions prepares to commit. If the local transaction commits first then the central transaction does not need to go into the authentication phase with the local sites to find out that it needs to abort. (It gets aborted by the concurrency control manager at the central site upon requesting authentication.) On the other hand, if the central transaction requests commit first then the authentication phase would be entered before the conflict was discovered. Let us focus on the conflict between a central (say new) and a local transaction. To be in conflict, their lock hold times on the same data item must overlap. Let us suppose that, on the average, a central lock is held between times $(0, R_{C1}/2)$. (Note that the average lock holding time is one half of the execution time.) In order to complete first, the local transactions must request a lock on the same data item anytime between $(-R_L/2, (R_{C1} - R_L)/2)$, assuming a lock hold time of $R_L/2$. This "winning" interval is of length $R_{C1}/2$. Otherwise the central transaction finishes first. Approximately, the probability of a local transaction finishing first is the probability P_{GL1} and is the ratio of the lock hold time of the central transaction to the sum of the lock hold times of the local and central transactions, respectively. In order to estimate the probability we will assume that the granule hold times of the two transactions are independent and thus write:

$$P_{GL1} = \frac{\frac{R_{C1}}{2}}{\frac{R_L}{2} + \frac{R_{C1}}{2}}, \tag{8}$$

$$P_{GL2} = \frac{\frac{R_{C2}}{2}}{\frac{R_L}{2} + \frac{R_{C2}}{2}}. \tag{9}$$

Now let us consider finite communications delay R_{COM}. The effect on (8) and (9) is similar to that on (3) and (4). Hence, (8) and (9) now become

$$P_{GL1} = \frac{\frac{\beta_1}{2}}{\frac{R_L}{2} + \frac{\beta_1}{2} + 2R_{COM}}, \tag{10}$$

$$P_{GL2} = \frac{\frac{\beta_2}{2}}{\frac{R_L}{2} + \frac{\beta_2}{2} + 2R_{COM}}. \tag{11}$$

Note that the $2R_{COM}$ term is only added to $\beta_1/2$ or $\beta_2/2$ in the denominator but not in the numerator. This is due to the fact that the communications delay increases the central transaction abort rate, but does not change the rate at which aborts are detected at the central site. At the end of the execution of a central transaction, there is still an R_{COM} interval of update information that is in transit from the local sites to the central site. Furthermore, before the authentication request arrives at the local site, there is another R_{COM} interval of updates occurring at the local sites. The conflicts occurring in the $2R_{COM}$ interval can only be found out after the authentication.

In order to evaluate the local transaction response times, transaction arrivals are modeled by a Poisson process. The expected total local transaction response time R_{TL} is expressed in terms of its components as follows:

$$R_{TL} = R_{CPU.L} + R_{IO.L} + R_{CONT.L}. \tag{12}$$

Each component will be described separately.

$R_{CPU.L} = R_{CPU.L}^{INPL} + R_{CPU.L}^{PROC}$ is the total time the transaction spends at the CPU, including both the CPU service and queueing times. The instructions executed by the transaction are divided into those for database calls associated with $R_{CPU.L}^{PROC}$ (10 calls per transaction with 25K instructions per call) and those for message processing and transaction initiation, associated with $R_{CPU.L}^{INPL}$ (150K instructions per transaction). Lock requests and database calls are assumed to occur uniformly over this transaction path length, breaking the transaction into many small tasks of equal size, each of which has to queue and be serviced by the CPU. From the trace analysis the average number of lock requests was found to be 15 per transaction. In addition, each lock/unlock request entails additional processing (2K 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. The average number of database I/O's per transaction for a single system was found to be 11, and an overhead of 3K instructions per I/O is used. CPU service time and queueing time is evaluated by modeling the CPU as an M/M/1 queue.

$R_{IO.L}$ 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 (or receipt of a message from the remote system processing the remote lock request). Thus,

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

where t_{IO} is the expected 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 expected number of I/O's per transaction, and consists of two kinds of I/O's. n_{IOPL} is the expected 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 expected 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 single system trace analysis, the average transaction performed 11 database I/O.

$R_{CONT.L}$ is the time spent in contention wait for a lock that is held by another transaction. The contention wait is estimated as

$$R_{CONT.L} = N_L \left(P_{LOC.LOC} \frac{R_L}{F} + P_{LOC.CEN}^{COHER} \frac{R_{HOLD}}{2} \right) \tag{13}$$

where N_L is the expected number of lock requests per transaction derived from trace analysis. $P_{LOC.LOC}$ is the probability of contention on a lock request with another transaction that runs locally, $P_{LOC.CEN}^{COHER}$ is the probability of contention on a lock request with a central transaction that holds locks at the local site during its authentication phase. The manner in which $P_{LOC.LOC}$ and $P_{LOC.CEN}^{COHER}$ are projected from the value measured in the trace driven simulation was described earlier in this section. R_L/F is the expected time a transaction waits if it contends with another local transaction for a lock, and is a function of the local transaction response time. This wait time is estimated as $(R_{TL} - R_{IOPL} - R_{CPU.L}^{INPL})/F$, where F is estimated below. The reason R_{IOPL} and $R_{CPU.L}^{INPL}$ are subtracted from the response time is because the transaction does not hold any locks during these times. $R_{HOLD}/2$ is the expected residual lock holding time for the centrally executing transaction during its authentication phase. R_{HOLD} is the amount of time a central transaction holds locks at the local site, during its authentication phase. Substituting these estimates for $R_{CONT.L}$ in (12) gives the total local transaction response time.

$$R_{TL} = R_{IOPL} + R_{CPU.L}^{INPL} + \frac{R_{CPU.L}^{PROC} + R_{IODB} + N_L P_{LOC.CEN} R_{HOLD}/2}{1 - \left(\frac{P_{LOC.LOC} N_L}{F} \right)}. \tag{14}$$

The reciprocal in the denominator of (14) indicates the expansion in the response time due to lock contention wait and is referred to as the lock contention expansion factor in [25].

The factor F for the fraction of the response time that represents the expected waiting time is estimated as follows. We neglect the probability of restart due to deadlocks. The mean wait time of a transaction that contends for lock is the mean remaining time of the transaction that it contends with. It is assumed that the transaction makes lock requests evenly over its execution. Thus, there are an equal number of transactions (including waiting and running transactions) that hold different numbers of locks. The probability of contention with a transaction is proportional to the number of locks that a transaction holds. Using a continuous time approximation, the mean remaining time r is

$$r = \int_0^{\bar{R}} P(x)(\bar{R} - x)\, dx,$$

where $P(x)$ is the probability of contention with a transaction that has run for time x, and $\bar{R}$ is the transaction response time, during which the transaction holds locks. Using the continuous approximation that a transaction holds a linearly increasing number of locks with time, $P(x)$ equals $x/\int_0^{\bar{R}} y\,dy$, which gives

$$r = \frac{\int_0^{\bar{R}} x(\bar{R} - x)\,dx}{\int_0^{\bar{R}} x\,dx} = \frac{\bar{R}}{3}. \tag{15}$$

Thus, F in (14) is estimated as 3. Note that if all lock requests are made at the beginning of the transaction then F equals 2.

The response time for the two coupled 14 MIPS systems at 20 transactions per second and contention probability of 0.0038 (as measured from the trace) was derived to be 0.65 seconds, using an equation similar to (14). In the approximate model, the feedback effect of the contention probability varying with response time is estimated using an iteration. The transaction response time is first derived for the initial estimate on contention probability. Then the resulting response time is used to compute a new contention probability explained previously in this section. The approximate model is then run again with the new contention probability. The iteration is repeated until convergence is obtained. Only a few iterations are required in the contention range considered.

The total response time of the transaction running at the central site R_{TC} is derived as follows:

$$R_{TC} = R_{CPUL}^{SHIP} + R_{COMDLY} + R_{CPUC}^{INPL} + R_{IOPL} + R_{C1} + \frac{p_A}{1 - \tilde{p}_A} R_{C2}. \tag{16}$$

R_{CPUL}^{SHIP} is the component of the response time at the local site at which the transaction arrives, and includes an overhead to determine that the transaction is of class B (20K instructions) and the communications overhead to ship the transaction to the central site (10K at each of the local and central sites), and at the end of the transaction to ship the results back from the central site (10K at each site). R_{COMDLY} is the communications delay involved in sending the transaction to the central site, and that to send the results back to the local site. It is estimated as $2R_{COM}$. The next three terms cover the time to run a new transaction, while the next to last term accounts for the expected re-run time. Analogous to the development of the response time expressions for the local transaction we write

$$R_{C1} = R_{CPU.C}^{PROC} + R_{IODB} + R_{CONT.C} + \gamma_1. \tag{17}$$

(Note that β_1 is the sum of the first three terms.) $R_{CPU.C}^{PROC}$ has components for transaction and commit processing. The instructions to execute the transaction (for the first and for reruns due to aborts) are the similar to that for the local transaction described above. At the central site the two phase commit processing overhead includes a constant overhead for each phase (1.5K instructions) plus an overhead for each local system involved in the commit operation (2K instructions per system in each phase), and communications overhead; at each local site involved in the commit there is communications overhead, an overhead for authentication (2K instructions), and an overhead to update the local database. As before, an M/M/1 model is used to estimate $R_{CPU.C}^{PROC}$. R_{IODB} has a component at the central site that is the same as that for the local transaction, and a component for updating the local database at commit time. $R_{CONT.C}$ is the expected wait time due to contentions with the other central transactions. When the communication delay is small compared to the local transaction response time, the aborted central transaction needs a back off wait time before reruns if an abort is detected at the distributed sites while the locks are still held by the local transactions. An additional wait time would be added to (17).

We next provide an estimate for $R_{CONT.C}$.

$$R_{CONT.C} = P_{CEN.CEN1} N_L w_1 + P_{CEN.CEN2} N_L w_2 \tag{18}$$

where w_1 (respectively, w_2) is the expected waiting time for a conflict with a first run (respectively, rerun) central transaction and is derived below. $P_{CEN.CEN1.}$ and $P_{CEN.CEN2.}$ are estimated previously in this section, and are the lock contention probabilities with new and rerun transactions at the central site, respectively. w_i can be estimated as follows.

$$w_i = \frac{\beta_i/2}{\beta_i/2 + \gamma_i}(\beta_i/3 + \gamma_i) + \frac{\gamma_i}{\beta_i/2 + \gamma_i}\gamma_i/2$$

$$i = 1, 2.$$

Note that as locks are requested uniformly over the transaction execution phase (β_i), the expected wait time for contention with a central transaction during this phase is $\beta_i/3 + \gamma_i$ (see derivation in (15)). As $\lambda N_{DS}(1 - p_{LOC}) N_L(\beta_1/2)$ is the expected number of locks held by first run central transaction during its execution phase (Little's formula) and $\lambda N_{DS}(1 - p_{LOC}) N_L \gamma_1$ is the expected number of locks held by first run central transaction during its authentication phase, and probability of contention with a first run central transaction during its execution phase is $\beta_1/2/(\beta_1/2 + \gamma_1)$, given that the transaction is contending with a central transaction in its first run. Similarly, the probability of contention with a first run central transaction during its authentication phase is $\gamma_1/(\beta_1/2 + \gamma_1)$, given that the transaction is contending with a central transaction in its first run. The expression for w_1, the expected waiting time for a conflict with a first run central transaction, follows directly from the above explanation. Similarly, w_2 can be derived.

Similar equations can be written for R_{C2}. The equations developed above have been encapsulated in an iterative procedure that estimates the transaction response time for central transactions.

As mentioned earlier, the models for the fully distributed and the centralized systems is the same as that described in [9]. We now sketch the principal differences of the above model for the hybrid system from those for the purely distributed and centralized systems. In the purely distributed system, both class A and class B transactions run at the local sites, with class B transactions making remote calls to access data. Locks are assumed to be acquired at the system from which the data is obtained. Therefore, there are no transaction aborts, and all contentions translate to lock waits. Similar estimates to the above for lock contention are derived for this case. In particular, (1)-(5) still apply, with *LOC* referring to Class A transactions and *CEN* to Class B transactions, and $p_A = 0$. A communication overhead and delay is added for each remote access of a class B transaction. The rest of the development is analogous to the above. For the centralized system, communication overhead and delay is added to each class A and class B transaction for communication of the input to the central site, and of results back from the central site. All transactions are executed at the central site, and again (1)-(5) apply with the same interpretations as that for the purely distributed system described above. Again, all contentions lead to waits for the required lock to be released with wait times estimated as for the hybrid system, and an analogous development is used to estimate overall response times.

IV. Results and Discussion

A. Simulation

The protocols were simulated using a discrete event simulation. Simulation results are presented for a 10 site distributed system, with each local site connected to the central site through a link with 200 msec communications delay; the central CPU has 10 MIPS while each local site has 1 MIPS. Transactions arrivals are Poisson processes, with the same arrival rate at each distributed site. The probability of local 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 (*lockspace*) is used. For local transactions, each local site makes lock requests uniformly over one tenth of the lock space, while central transactions make lock requests uniformly over the entire lock space. In analyzing this case the constant C of Section III-B-3, is estimated as $C = 1/lockspace$. The model 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. In the case of a contention that leads into a deadlock the transaction is aborted and all locks held are released. To reduce simulation times, the commit processing is simplified; specifically, I/O delays for update of the local site from the central site and the second commit phase are not simulated.

We now compare results of the simulation and analysis for the above parameters. Fig. 2 shows the probability of the first abort versus total transaction rate for central

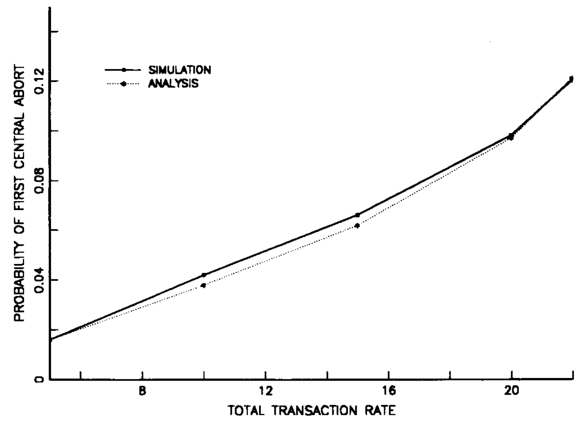


Fig. 2. Simulation and analysis estimates of abort probability.

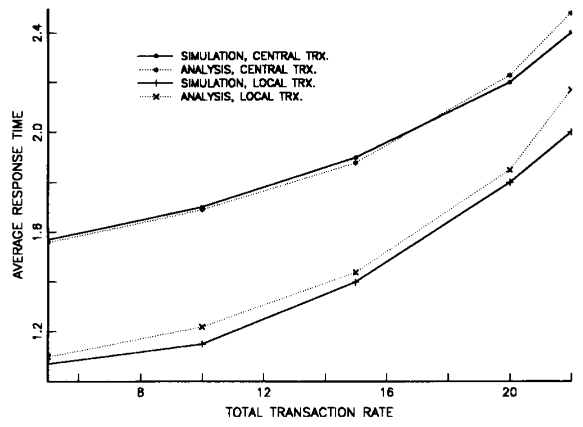


Fig. 3. Simulation and analysis estimates of response time.

transactions. The estimates for the simulation and analysis are very close. Fig. 3 shows the average local and central transaction response times versus total transaction rate. The difference between analysis and simulation is within 5% for processor utilizations of less than about 60%, and is within 10% for higher utilizations.

B. Performance Comparison

We now present some further projections from the analytic model. We begin by illustrating the tradeoffs between the approaches for the case of 16 geographically distributed sites. We examine the effect of changing the mix of transaction types (class A versus class B transactions), varying the communication delay and the transaction rate. Initially we assume that strict consistency is required (as in the scheme described in Section II). We also assume that class B transactions access data controlled by different distributed systems for each database call (i.e., worst case), and hence the number of systems in the commit processing for class B transactions equals the number of database calls. Later, we examine the effect of relaxing these assumptions.

Fig. 4 shows the average response time versus fraction

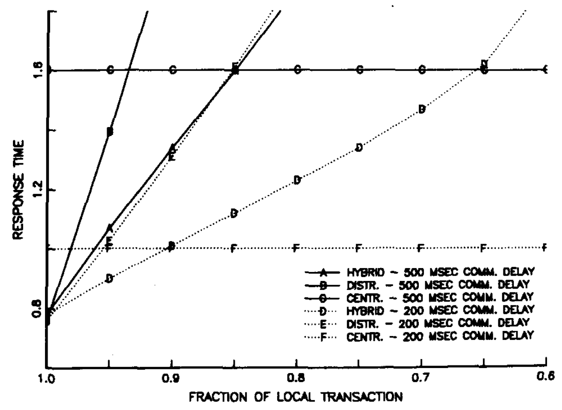


Fig. 4. Response time of different approaches.

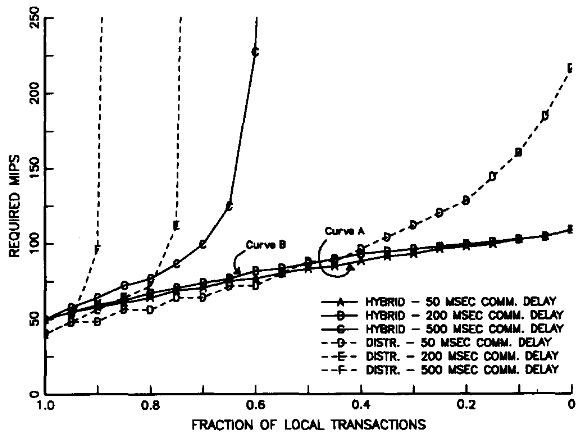


Fig. 5. MIPS required versus locality.

of class A transactions (i.e., fraction of local transactions), for the different approaches. The graph is for a total of 80 MIPS. For the centralized system we assume a single large processor, while for the distributed system each of the 16 processors is 5 MIPS. For the hybrid system, the MIPS at the central site and the distributed sites are adjusted so as to minimize the response time, while constraining the total to be 80 MIPS. The optimal MIPS at the central and local sites for the hybrid system depend on the fraction of class A transactions. For example, with 60% class A transactions, the optimal configuration has about 37 MIPS at the central site, and 43 MIPS distributed among the distributed sites; with 80% class A transactions, the optimal hybrid system configuration has about 21 MIPS at the central site and 59 MIPS distributed among the distributed sites. Three values for the communications delay are examined, where the communications delay is the difference in the times when a message is sent out from a system and when it is received by the destination system. We assume that this delay is the same for sending a message between distributed systems and between a distributed and the central system. First consider a communications delay of 500 ms. The chart indicates that the average response time for the distributed and hybrid systems is lower than that of the centralized system if the fraction of class B transactions is small. For the distributed system, if the percentage of class B transactions is greater than about 5%, the average response time projected is worse than that of the centralized system. The average response time of the hybrid system is better than that of the centralized system until the percentage of class B transactions is greater than about 15%. This indicates that the hybrid system is significantly more stable with respect to the fraction of class B transactions than the fully distributed system. The chart also shows a similar comparison for a communications delay of 200 ms. The figure indicates that as the communications delay decreases, the fraction of class B transactions at which the crossover in average response times occurs decreases. For a 50 ms communications delay (not shown in the figure), the centralized system's average response time is better than the distributed and hybrid systems even when all transactions are local. This occurs because the centralized system is assumed to consist of one large processor, and the larger communications time required by the centralized system is balanced by smaller processing times. As we see later, if the consistency requirement between local and central data is relaxed, the hybrid system can do substantially better than the centralized system over a wider range of parameter values.

In Fig. 5 we examine the MIPS required to support a total transaction rate of 80 transactions per second, while providing an average response time of not more than two seconds. Again, for the hybrid system, the distribution of MIPS between the central site and distributed sites is chosen so as to minimize the average response time obtained. The chart shows the MIPS required versus the fraction of class A transactions, for different values of the communications delay. The chart indicates that the distributed system is rather sensitive to the fraction of class B transactions, while the hybrid system is much less sensitive to this factor. Further, for larger communications delays, the distributed system is even more sensitive to the fraction of class B transactions, while the hybrid system is significantly less sensitive to this parameter for higher values of the communications delay. Note, however, for very small fractions of class B transactions, the MIPS required for the distributed system is slightly less than that for the hybrid system. This is because the hybrid system consumes some MIPS in (asynchronously) sending update information to the central site, and in updating replicated data at the central site.

We next examine the MIPS required to support a varying transaction rate, with an average response time bound of two seconds. Fig. 6 shows the MIPS required versus total transaction rate, assuming 25% of class B transactions. Three values of communications delay, 50, 200, and 500 ms, are examined. Observe that the distributed system is more sensitive to the communications delay than the hybrid system. For a small communications delay (e.g. 50 ms), as the transaction rate increases, the MIPS

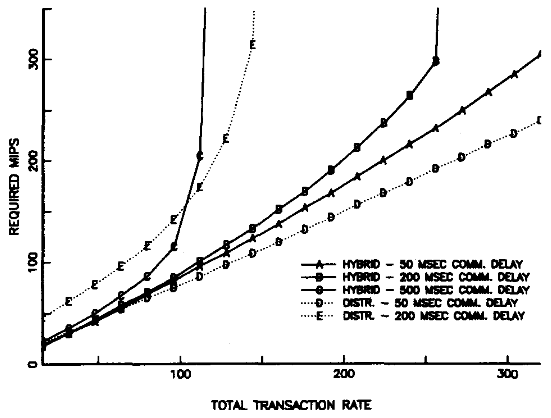


Fig. 6. MIPS required versus transaction rate.

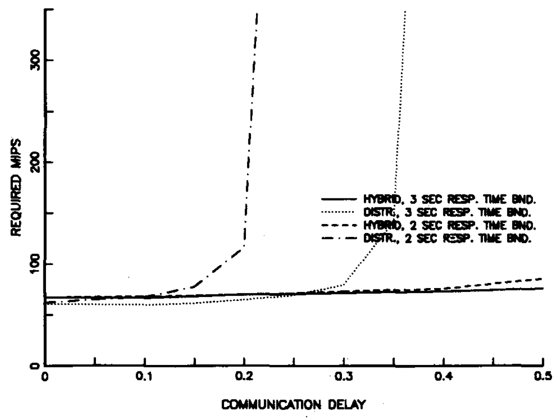


Fig. 7. MIPS required versus communication delay.

required for the distributed system is a little lower than that for the hybrid system. The reason for this is that, for the hybrid system protocol described, some central site MIPS are consumed in aborts due to contention with transactions running at the local sites, and the transaction abort probability increases with the total transaction rate. At higher communications delays, the hybrid system has a lower MIPS requirement for all transaction rates. This is because the larger communications overheads and delays of the fully distributed system outweigh the increase in abort overheads and communications for the hybrid system. The distributed system cannot satisfy the two second response time bound with a communication delay of 500 ms; hence this curve is absent from the figure.

In Fig. 7 we examine the MIPS required as a function of the communications delay. The chart is for a total transaction rate of 80 transactions per second, 25% of class B transactions, and an average response time bound of 2 seconds and 3 seconds. Notice again, that the MIPS required by the hybrid system is much less sensitive to communications delay than that for the distributed system. This is because the hybrid system is able to meet the response time bound with fairly large communications delay, while the distributed system cannot.

In the above charts we have assumed that for class B transactions, $(n - 1)/n$ of the database calls are to data located in systems other than the system at which the transaction originated. In Fig. 8, we examine the effect of locality of the database calls for class B transactions. The abscissa in this chart is the probability that a database call for a class B transaction is local to the originating system. The ordinate is the required MIPS to satisfy a 2 second average response time bound, for 16 sites, with a total of 80 transactions per second, and assuming 75% class A transactions. The parameter is the communications delay. The chart indicates that when the class B transactions are largely local (i.e., make few calls to the remote database), the distributed system requires fewer MIPS than the hybrid system. The cross-over point between the two approaches depends on the communications delay. This

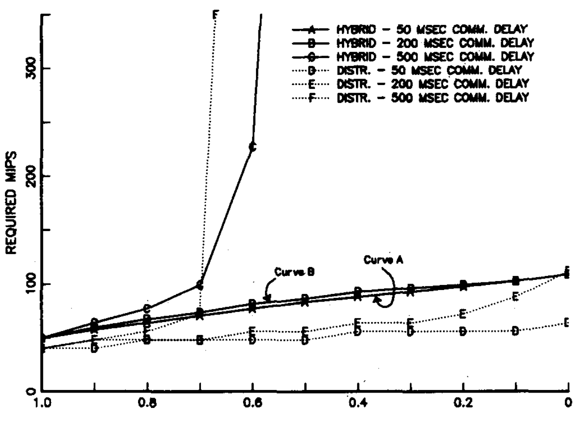


Fig. 8. Effect of locality of class B transactions.

suggests the introduction of a third class C of transactions, for which there are few (average) remote calls per transaction. The hybrid system could run such class C transactions on the local systems (rather than shipping them to the central system), making remote database calls and doing two phase commits as in the distributed system. The concurrency and coherency protocol for the hybrid system with class C transactions is essentially the same as before, since for concurrency and coherency the class C transactions obtain local locks and are similar to the class A transactions for this purpose.

We now examine the effect of relaxing the consistency requirement. We consider the case when a fraction of the transactions of each class are read only transactions. The parameters for the read-only transactions are assumed to be similar to the ordinary transactions, except that they have no updates and consequently have three fewer database calls and three fewer I/O's on the average per transaction. Further, we assume that a read only transaction shipped to the central site, can read data that is concurrently updated by a transaction running at a local system. Thus, the coherency control requirement is relaxed and we allow a read only transaction running at the central site to commit without communicating with the local sites. In terms of contention, with the above assumption read only transactions do not have local-central contention. For the distributed system, with the above assumptions, there is no need for a two phase commit protocol for read only transactions. Fig. 9 shows the number of MIPS required for the distributed and hybrid systems versus the fraction of read-only transactions, for 16 systems with a 2 second response time bound, a total transaction rate of 80 transactions per second, and 75% class A transactions. For the hybrid system, the MIPS required for all read only transactions is approximately one-half the MIPS required for no read-only transaction. This is due to reduction in transaction pathlength, elimination of overheads for propagating updates at local sites to the central site, and elimination of the coherency control protocol for class B transactions. In addition, the read only transactions have no aborts and no lock contention, further reducing the MIPS requirement. The MIPS requirements under the three communication delays, 50, 200, and 500 ms, are roughly the same as the fractions of read-only transactions vary. For the distributed system the MIPS required decreases primarily due to the elimination of commit overheads and delays. The effect is more pronounced for larger communications delays. This is because, for larger communications delays, meeting the response time constraint of 2 seconds becomes difficult. Still there is no curve for a 500 ms communications delay. Since the commit delays are eliminated, the response time constraint is met by allowing larger CPU portion of the response time, leading to a larger CPU utilization and fewer MIPS required.

In Fig. 10 we examine the sensitivity to the base contention observed in the trace driven simulation, from which the contention for the above cases was derived. The figure shows the response time versus the fraction of class A transactions. The parameter is the base contention probability which is varied from zero to one and a half that observed in the traces. The hybrid system shows more sensitivity to the contention probability than the distributed system. The distributed system is affected primarily by the communications delay, rather than by contention. In the hybrid system, the probability of aborts of class B transactions increases with contention. Even so, for reasonable variation of the base contention probability around that measured in the trace, the above conclusions are not affected.

Next we examine the effect of varying the number of systems, while keeping the total transaction rate fixed. Fig. 11 shows the MIPS required versus the fraction of class A transactions, for a total of 80 transactions per second, an average response time constraint of 2 seconds, and a communications delay of 500 ms. The parameter is the number of distributed systems. The graph indicates that for the same fraction of class A transactions, there is a small change in the total MIPS required due to variation in the number of distributed systems. The MIPS increase for larger number of distributed systems is primarily due to the fact that more systems need to be coordinated during commit. However, it should be noted that generally speaking, the fraction of local (class A) transactions may decrease as the number of systems increases; precisely how this fraction changes will depend on the environment and the locality for a particular application.

In the above, we have assumed that for class B transactions the number of systems in commit processing is the

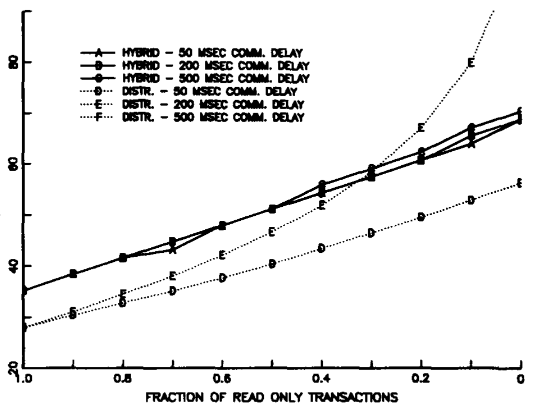


Fig. 9. Effect of relaxed coherency control.

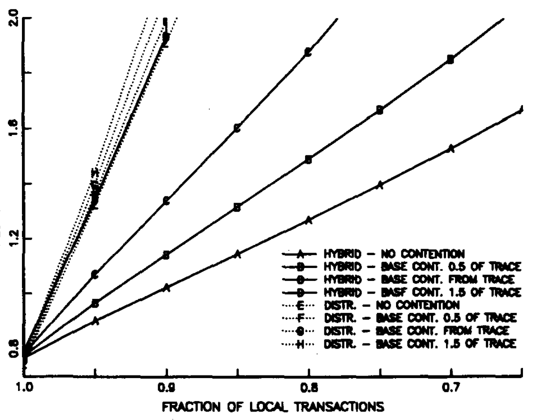


Fig. 10. Sensitivity to base contention probability.

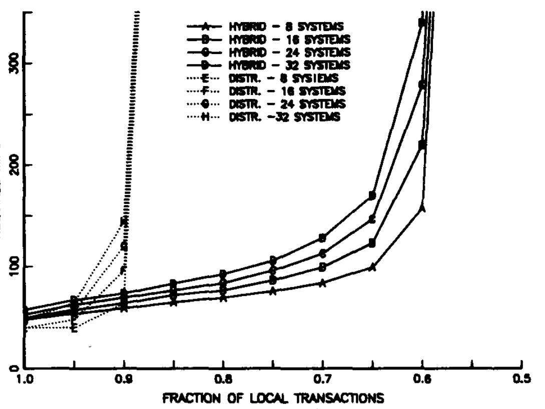


Fig. 11. Effect of all remote calls to one system.

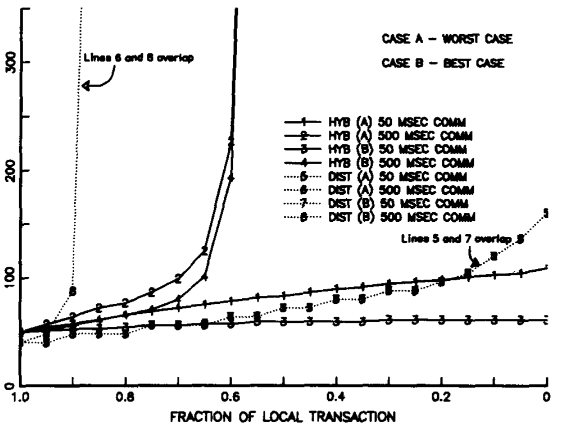


Fig. 12. Effect of varying number of systems.

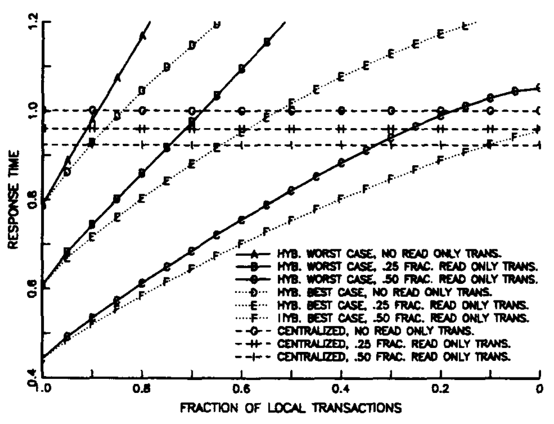


Fig. 13. Response time comparison with read only transactions.

number of database calls (i.e., all remote database calls are to different systems). In Fig. 12 we compare this with the best case when all remote database accesses are to the same system. The chart shows the MIPS required versus the fraction of class A transactions and is for 16 systems, a total transaction rate of 80 transactions per second, and an average response time constraint of 2 seconds. While the chart shows some decrease in the required MIPS for the hybrid case when all remote database calls are to the same system, it does not affect the conclusions drawn above.

In Fig. 4, we had observed that as the communications delay decreases, the cross-over of response time for the hybrid and centralized approaches occurs at smaller values of the fraction of remote transactions. We now examine the effect of relaxed consistency requirement, and of all remote calls directed to the same system, on this comparison. Fig. 13 shows the response time versus the fraction of class A transactions for the centralized system and hybrid system, for different assumed fractions of read only transactions. As before, we assume that the coherency control requirement is relaxed for read-only transactions; thus, read only transactions shipped to the central site can read data that is concurrently updated by a transaction running at a local system. The curves labeled as "worst case" assume that all remote calls are directed to different systems, while the curves labeled as "best case" assume that all remote calls are to the same system. The curves labeled as "worst case," with no read-only transactions, are the same as those in Fig. 4, for 200 ms communications delay. The cross-over in these curves is at about 10% class B transactions. When all remote calls are to the same system (best case) with no read-only transactions, the cross-over in the curves moves to about 15% class B transactions. If 25% of the transactions are read-only, then the cross-over for the worst and best cases move to about 30% and 40% class B transactions respectively. For 50% read only transactions, the cross-over for the worst and best case are at about 70% and 90% class B transactions, respectively.

V. Conclusions

In this paper we examined a hybrid system structure consisting of distributed computer systems connected to a central computing complex. We described a protocol for concurrency control for transactions running at the local and central systems, and for maintaining coherency of data replicated at the central and distributed systems. The protocol ensures consistency using a combination of optimistic and pessimistic methods. The protocol allows transactions that access purely local data at a local system to run at a distributed system and to commit without any communications to the central site. Updates at a distibuted system are asynchronously propagated to the central site. Transactions that access nonlocal data, are shipped to the central site for execution. Data is replicated at the central and distributed sites so that transactions running at the central site need communicate with the distributed systems only at commit time, allowing for good performance for transactions running at the central site. In this manner, the hybrid system offers the advantage of distributed systems for transactions that access local data, and the advantage of centralized systems for transactions that access nonlocal data.

A performance model for the hybrid system was developed. The centralized, distributed, and hybrid system approaches were compared. It was found that the distributed system is very sensitive to the fraction of transactions that access remote data. The hybrid system is much less sensitive to the fraction of remote accesses. This effect is more pronounced for larger communications delays between systems. The total MIPS required by the hybrid and distributed systems to satisfy a response time constraint was compared. This MIPS comparison depends on the fraction of remote access, communications delay and transaction rate. The distribution system requires slightly lower total MIPS than the hybrid system if the fraction of transactions with remote calls and the communications delay is small, but requires larger MIPS otherwise. The MIPS required was also compared for a varying transaction rate, at a fixed (25%) nonlocal transaction fraction;

the distributed system required slightly smaller MIPS at small communications delay, but much larger MIPS as the communications delay increased. The effect of relaxing the consistency requirement was examined and was found to significantly reduce the MIPS requirement. The sensitivity to the (base) lock contention, and the number of distributed systems was examined; it was found that varying these parameters did not change the above conclusions. In this paper we have focused on the performance of the hybrid system and compared it with the fully distributed and centralized system approaches. A comparison of the reliability characteristics of these approaches is a topic for further work.

REFERENCES

[1] D. Barbara and H. Garcia-Molina, "How expensive is data replication? An example," in *Proc. 2nd Distributed Computing Systems*, Feb. 1982, pp. 263-268.

[2] 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.

[3] B. Ciciani, D. M. Dias, and P. S. Yu, "Load sharing in hybrid distributed-centralized database systems," in *Proc. 8th Int. Conf. Distributed Computing Systems*, San Jose, CA, June 1988, pp. 274-281.

[4] 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. 78-81, Jan. 1981.

[5] 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.

[6] D. Asit, 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.

[7] C. J. Date, *An Introduction to Database Systems*, vols. 1 and 2. Reading, MA: Addison-Wesley, vol. 1, 1981, vol. 2, 1983.

[8] D. M. Dias, B. R. Iyer, and P. S. Yu, "Tradeoffs between coupling small and large processors for transaction processing," *IEEE Trans. Comput.*, vol. 37, no. 3, pp. 310-320, Mar. 1988.

[9] D. M. Dias, P. S. Yu, and B. T. Bennett, "On centralized versus geographically distributed database systems," in *Proc. 7th Int. Conf. Distributed Computing Systems*, Berlin, West Germany, Sept. 1987.

[10] B. I. Galler and L. Bos, "A model of transaction blocking in databases," *Perform. Eval.*, vol. 3, pp. 95-122, 1983.

[11] J. Gray, P. Homan, R. Obermarck, and H. Korth, "A straw man analysis of probability of waiting and deadlock," IBM, San Jose, CA, Res. Rep. RJ 3066.

[12] J. N. Gray, "An approach to decentralized computer systems," *IEEE Trans. Software Eng.*, vol. SE-12, no. 6, pp. 684-692, June 1986.

[13] IBM Corporation, *Customer Information Control System Virtual Storage (CICS/VS): Intercommunication Facilities Guide*, SC33-0133, 1983.

[14] 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 of Data*, Boston, MA, Jan. 1979, pp. 134-142.

[15] J. A. Larson and S. Rahimi, *Tutorial: Distributed Database Management*. Silver Spring, MD: IEEE Computer Society Press, 1985.

[16] S. Lavenberg, "A simple analysis of exclusive and shared lock contention in a database system," *Perform. Eval. Rev.*, (Proc. 1984 ACM SIGMETRICS Conf., vol. 12, no. 3, pp. 143-148.

[17] D. Mitra and P. J. Weinberger, "Probabilistic models of database locking: Solutions, computational algorithms and asymptotics," *J. ACM*, vol. 31, no. 4, pp. 855-878, Oct. 1984.

[18] R. D. Nelson and B. R. Iyer, "Analysis of a replicated data base," *Perform. Eval.*, vol. 5, pp. 133-148, 1985.

[19] D. Potier and P. Leblanc, "Analysis of locking policies in database management systems," *Commun. ACM*, vol. 23, no. 10, pp. 584-593, Oct. 1980.

[20] J. P. Strickland, P. P. Uhrowczik, and V. L. Watts, "IMS/VS: An evolving system," *IBM Syst. J.*, vol. 21, no. 4, pp. 490-510, 1982.

[21] Y. C. Tay, "A mean value performance model for locking in databases," Ph.D. dissertation, Harvard Univ., Cambridge, MA, Feb. 1984.

[22] 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, Israel, June 1982), P. Scheuermann, Ed. New York: Academic, 1982, pp. 1-27.

[23] P. S. Yu, D. M. Dias, J. T. Robinson, B. R. Iyer, and D. Cornell, "Modeling of centralized concurrency control in multi-system environment," *Perform. Eval. Rev.*, (*Proc. 1985 ACM SIGMETRICS*), vol. 13, no. 2, pp. 183-191.

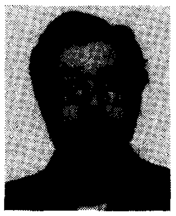
[24] P. S. Yu, D. W. Cornell, D. M. Dias, and A. Thomasian, "Performance comparison of IO shipping and database call shipping: Schemes in multisystem partitioned databases," *Perform. Eval.*, vol. 10, pp. 15-33, 1989.

[25] 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.

[26] P. S. Yu and D. M. Dias, "Impact of large memory on the performance of optimistic concurrency control schemes," in *Proc. PARABASE-90: Int. Conf. Databases, Parallel Architectures and their Applications*, Miami Beach, FL, Mar. 1990, pp. 86-90.

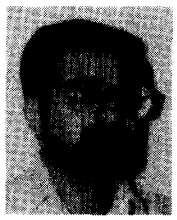
[27] ——, "Notes on modeling optimistic concurrency control schemes," IBM, Yorktown Heights, NY, Res. Rep. RC 14825, Aug. 1989.

[28] P. S. Yu, D. M. Dias, and S. S. Lavenberg, "On modeling database concurrency control," IBM, Yorktown Heights, NY, Res. Rep. RC 15386, Jan. 1990.



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 Ingegnerial Elettronica, University of Rome (Tor Vergata). From 1987 to 1988 he was a Visiting Scientist at the IBM Thomas J. Watson 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, India, 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, Yorktown Heights, NY. His current research interests include database systems, transaction and query processing, parallel and distributed systems, interconnection networks, and performance analysis.

Balakrishna (Bala) R. Iyer received the B.Tech. degree from the Indian Institute of Technology, Bombay, India, and the M.S. and Ph.D. degrees from Rice University, Houston, TX.

He has previously worked for Bell Telephone Laboratories, Murray Hill, NJ, as a Member of Technical Staff. He joined IBM in 1982 as a Research Staff Member with the Architecture and Analysis Group of the IBM Thomas J. Watson Research Center, Yorktown Heights, NY. Since May 1989 he has been with the Data Base Technology Institute at the IBM Santa Teresa Laboratory, San Jose, CA. His research interests include database systems, data compression, computer architecture, performance and reliability analysis, fault-tolerant computing, voice-data integration, computer networks, and videotex systems.

Philip S. Yu (S'76-M'78-SM'87) received the B.S. degree in electrical engineering from National Taiwan University, Taipei, Taiwan, Republic of China, in 1972, the M.S. and Ph.D. degrees in electrical engineering 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, which focuses on the design and analysis of clustering multiple processors for transaction and query processing. His current research interests include database management systems, parallel and distributed processing, computer architecture, performance modeling, and workload analysis. He has published about 70 papers in refereed journals and conferences and has published over 60 research reports and 50 technical disclosures.

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