Analysis of Concurrency–Coherency Control Protocols for Distributed Transaction Processing Systems with Regional Locality

Bruno Ciciani, Daniel M. Dias, and Philip S. Yu, *Senior Member, IEEE*

Abstract— **In this paper we examine a system structure and protocols to improve the performance of a distributed transaction processing system when there is some regional locality of data reference. Several transaction processing applications such as reservation systems, insurance, and banking belong to this category. While maintaining a distributed computer system at each region, a central computer system is introduced with a replication of all databases at the distributed sites. It can provide the advantage of distributed systems for transactions that refer principally to local data, and also can provide the advantage of centralized systems for transactions accessing nonlocal data. Specialized protocols can be designed to keep the copies at the distributed and centralized systems consistent without incurring the overhead and delay of generalized protocols for fully replicated databases. In this paper we study the advantage achievable through this system structure and the trade-offs between protocols for concurrency and coherency control of the duplicate copies of the databases. An approximate analytic model is employed to estimate the system performance. It is found that the performance is indeed sensitive to the protocol and substantial performance improvement can be obtained as compared with distributed systems. The protocol design factors considered include the approach for intersite concurrency control (optimistic versus pessimistic), resolution of aborts due to intersite conflict, and choice of the master/primary site of the dual copies (distributed site versus central site). Among the protocols considered, the most robust one uses an optimistic protocol for intersite control with the distributed site as the master site, allows a locally running transaction to commit without any communication with the central site, and balances transaction aborts between transactions running at the central site and distributed sites.**

Index Terms—**Analytic model, concurrency control, data replication, distributed database, performance analysis, transaction processing.**

I. Introduction

In a distributed database environment [17], [19] replication of databases has often been considered for either performance [21] or reliability reasons [16]. Although replication can improve the performance in a read-only transaction environment, overall response time can deteriorate in the presence of update transactions. This is especially the case when the database is fully replicated over all sites. In some application environments, such as airline reservation, banking, or insurance claims processing systems, there is significant regional locality of data reference. For instance, for a travel reservation system, transactions for each customer with regard to his travel plan and pricing mostly occur at the travel agency of the person's home/office location. Occasionally, during the travel, plans change because of weather or some other reason. Suppose that the organization has computing facilities not only at a central site but also at a few regional centers. Certainly, one place to keep a customer database record is the regional center where that customer resides. Another alternative place to keep the record is at the site of the central system. In [13] the trade-offs between a centralized database and a (fully) distributed database is studied where replication of databases is not considered. In these systems the databases are partitioned and distributed among regional processing systems, and a request shipping mechanism is provided to support the access of data in a nonlocal database partition [11], [18], [24]. Such a distributed system was found to be very sensitive to the fraction of nonlocal data references. Making full replication of databases at all regional centers in this case makes little sense as the update traffic is significant and there is a clear preference site of transaction origin for each customer, at the region of his home location. However, as we will demonstrate in this paper significant performance improvements can be obtained by maintaining a dual copy at the central system. This is referred to as a hybrid distributed-centralized database system structure [8], [9] since it essentially consists of a number of geographically distributed systems connected by a network to a central computing complex. The hybrid architecture can potentially provide the advantage of (geographically) distributed systems for transactions that refer principally to local data, and also provide the advantage of centralized systems for transactions that access a lot of nonlocal data. The additional copy at the central site can also enhance the availability as compared to a fully distributed system without replication.

The scenario considered is illustrated in Fig. 1. User workstations are assumed to be connected to geographically distributed computer systems. This may be through a local or metropolitan area network. The distributed systems in turn are connected to a central computing complex through a long-haul communications network. The data at each distributed system is replicated only at the central system; if data is not replicated at the central system, then a transaction running

Manuscript received May 21, 1991; revised March 26, 1992. Recommended by E. Gelenbe.

B. Ciciani is with the Computer Science Department, University of Rome, La Sapienza, Rome, Italy.

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

IEEE Log Number 9202820.

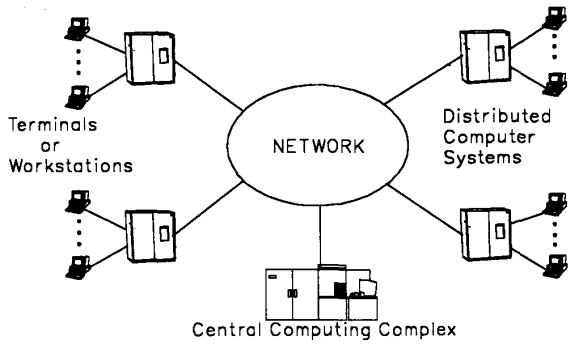

Fig. 1. Hybrid system structure.

at the central site would have to access remote data at the distributed systems, resulting in poor performance at the central site. In order to provide the advantage of the distributed systems for transactions that access local data only, it should be possible to commit such transactions without waiting for communication to the central site; otherwise, it would be preferable to ship the entire transaction to the central site eliminating the advantage of the distributed system. This implies that when data that is replicated at the central and distributed sites is updated at a distributed site, the update must be propagated asynchronously to the central site. Consequently, the data at the central and distributed sites are not always coherent, and concurrency–coherency protocols must ensure that transactions use consistent data. In this paper we identify the critical factors in the design space for the hybrid system to understand the impact of the protocol and to further explore the potential advantages of hybrid system alluded to in [8] where the trade-offs of different protocols are not addressed. The factors considered include the approach for intersite concurrency control (optimistic versus pessimistic), resolution of aborts due to intersite conflict, and the master/primary site of the dual copies (distributed site versus central site). We examine and analyze alternate protocols and the trade-off between the protocols. Among the protocols considered, the most robust one uses an optimistic protocol for intersite concurrent control with the distributed site as the master site, allows a locally running transaction to commit without any communication with the central site, and balances transaction aborts between transactions running at the central site and distributed sites.

Most previous analytical models of distributed databases with replication have been for very simple models. The models in [10], [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 [22], 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. In distributed database systems, most other previous analyses of concurrency control have either assumed no data replication [2], [20] or full replications [14], [15]. Simulation studies of partially replicated systems are reported in [3], [4], and [23]. In [8], the performance of the hybrid distributed-centralized system is considered under a specific protocol, referred to in this paper as the locally oriented protocol (see Section II). In [7], an analytic approach is used to study the optimal degree of replication in a distributed database environment. The methodology and analysis for data contention extend that from centralized database systems using locking reported in [6], [25], [26], and [28] and optimistic protocol in [27]–[29], and distributed databases without replication also using locking [13]. It is based on a decomposition principle introduced in [25], [28], and [29] to analyze data contention and hardware resource (e.g., CPU) contention separately and capture the interaction through an iteration. The methodology is quite general and can be employed to analyze the various protocols considered here for the hybrid system environment.

In Section II we explore the design space of the concurrency–coherency control protocol for the hybrid system and identify the critical design factors. We then describe the hybrid system protocols that we consider, and qualitatively discuss their potential. Section III presents the analysis. Comparison of the trade-offs between different protocols is given in Section IV. Concluding remarks appear in Section V. The Appendix provides details of the estimation of contention and abort probabilities.

II. Concurrency and Coherency–Control Protocols

There is a large design space of alternative concurrency and coherency–control protocols for the environment of Fig. 1. We will not focus on the concurrency–control protocol among transactions at the same site, since this is an orthogonal aspect, and has been studied extensively [1]; we assume that locking is used for this purpose. We consider the primary site approach, where either the distributed sites or the central site are master (primary site) for the partition of the data they have. For the primary site approach, the first issue is the location of the master site or primary copy. The choice of master site is indicated as the first choice in a tree of possible choices for protocol selection, as indicated in Fig. 2. The primary site approach has the advantage in this environment that, with the local sites as master, local transactions can be committed without any communication to other sites, as described below. For the concurrency control protocol between transactions running at different sites, either an optimistic or a pessimistic approach can be taken, as indicated in Fig. 2. The optimistic protocols across sites have the advantage that only a single set of communications at transaction commit time is necessary [4], [5], while the

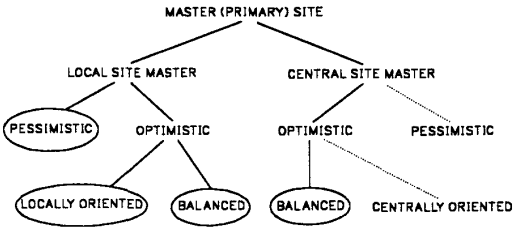

Fig. 2. Protocol alternatives considered.

pessimistic protocols require multiple communications during the execution of the transaction. For the optimistic protocols there is a further choice of which transaction to abort when conflicts are detected at commit time between transactions running at different sites. For instance (referring to Fig. 2) for the protocol with the local sites as master with an optimistic intersite concurrency control, the locally oriented protocol gives priority to transactions running at the distributed sites and only aborts conflicting transactions running at the central site at commit time. This policy is desirable if indeed it is required to provide better response time to transactions running locally at the distributed sites and accessing local data. Therefore, we refer to this protocol as the **locally oriented protocol** [8]. However, we will see that at high transaction rates it can result in excessive aborts of transactions running at the central site, so that the response time of transactions shipped to the central site becomes very large, while transactions running at the distributed sites are not affected. The **balanced protocols** indicated in Fig. 2, abort any conflicting running transaction when a commit request arrives at the master site. Thus the balanced protocols distribute the aborts due to conflicts between central and locally running transactions. For the case with the local sites as master we consider both the balanced protocol and the locally oriented protocol. As we will see, the case with the central site as master does not take advantage of transaction locality, and hence only the balanced case is shown to illustrate the point. For the purposes of comparison, a completely pessimistic protocol is also examined. The protocols that we compare in this paper are encircled in Fig. 2.

The following classes of transactions are identified. Some transactions only require local data, and do not need data from any other site (class A). The second class B of transactions usually require nonlocal data. We assume that with some preprocessing it is possible to identify the class of a transaction. Class A transactions will be run on the local systems, i.e., the distributed system local to the terminal entering the transaction, and any updates will be propagated asynchronously to the central site; class B transactions will be shipped in their entirety to the central site. Subsequently, class A (respectively class B) transactions will be referred to as **local** (respectively **central**) transactions.

We now describe the protocols that we will analyze and compare subsequently. We first describe the "balanced" protocols outlined above and then describe the differences of the "locally oriented" protocols from the balanced protocols.

Balanced protocol with local site as master: As stated above, the database is partitioned among the distributed systems, and is replicated at the central site. Each distributed system is concurrency–coherency 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 (share or exclusive mode). The lock manager maintains two fields for each lock—a concurrency control field (share 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 (see below) of updates being propagated to the central site, and is initially set to zero.

The relevant concurrency and coherency control actions at a local site are described in pseudocode form in Fig. 3. Locking is used for concurrency control among local transactions at the same site, and the actions for lock requests in Fig. 3 are the usual ones and are self-explanatory. At transaction commit point (command LOCAL_COMMIT in Fig. 3), first a check is made if the (locally running) transaction has been marked for abort (by a committing or committed central transaction), as explained below. If so, the class A transaction is aborted and restarted without releasing locks held by this transaction. (The rationale for continuing to hold locks acquired during the first run is based on the observation in [12], [28], and [29] that, with sufficient memory, the rerun transaction will find all the granules in memory, since it read the data during its first run. Thus, the second run of the transaction has no (or very few) I/O's and is therefore very short, making it profitable to continue to hold the local locks. Note that the updated version of the data is provided through the coherency control mechanism which has also marked the transaction for abort.) If the class A transaction is not marked for abort, the concurrency control field is used to release the share or exclusive lock held by the transaction (transactions queued on the entity can then be granted the lock); further, the coherence field count of the lock is incremented to indicate pending response from the central site. Asynchronously, a message is sent to the central site with the updated granules. 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 for 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 (though the transaction can commit) the asynchronous

```
local_cc_mgr(command,trx_id,gran_list,type)
{ /* local/distributed site concurrency and coherency manager actions */
  if(command = = LOCK) /* locking used for conc. control among local trans. */
  { if(compatible) /* compatible with current locks held */
    { grant lock ; return message(GRANTED) }
    else
    { if(deadlock) /* standard check for cycle of waiters at local site only */
      { release all locks held by transaction trx_id and grant locks to compatible waiters
        return message(ABORT) /* abort transaction */
      }
      else
      { enqueue ; return message(LOCK_CONFLICT_WAIT) } /* place on wait queue for granule */
    }
  }
  if(command = = LOCAL_COMMIT) /* commit of a locally running class A trans. */
  { if(transaction marked for abort) /* A central trans. in commit can mark a local trans. for abort */
    { release all locks held by transaction trx_id and grant locks to compatible waiters
      return message(ABORT)
    }
    else /* commit transaction */
    { for all granules held exclusive by transaction
      { increment coherence_count(granule)
        send central_cc_mgr(CENTRAL_UPDATE,trx_id,granule) /* asynch update to central site */
      }
      release all locks held by transaction trx_id and grant locks to compatible waiters
      return message(DONE)
    }
  }
  if(command = = CENTRAL_UPDATE_ACK) /* central site response to async. update */
  { decrement coherence_count(granule) }
  if(command = = CENTRAL_COMMIT_1) /* first phase commit of central trans. */
  { done = 0
    for all(granule in gran_list) while(done = = 0)
    { if(coherence_count(granule) != 0) /*some in-flight update */
      { send central_cc_mgr(COMMIT1,trx_id,,ABORT) ; done = 1 } /* abort trans. */
    }
    if(done = = 0) /* no in-flight updates */
    { for all(granule in gran_list)
      { if(granule held in incompatible mode)
        { mark holder of incompatible transactions for abort /* loc. or. prot. would abort central trans. */
          release incompatible locks held by aborted transactions
        }
        grant lock to transaction trx_id /* central trans. holds lock till committed */
      }
      write prepare to commit log /* can be done asynchronously outside CC manager */
      send central_cc_mgr(COMMIT1,trx_id,,ACK)
    }
  }
  if(command = = CENTRAL_COMMIT_2) /* second phase commit of central trans. */
  { write log /* can be done asynchronously outside CC manger */
    if(type = = COMMIT) { update database }
    release all locks held by transaction trx_id and grant locks to compatible waiters
    send central_cc_mgr(COMMIT2,trx_id,,ACK)
  }
}
```

Fig. 3. Pseudocode for distributed site concurrency–coherency control (balanced protocol).

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. The relevant actions of the central site concurrency–coherency controller are shown in pseudocode form in Fig. 4. At the central site, local locks are used for concurrency control among transactions running at the central site, with the corresponding actions in Fig. 4 being self-explanatory. However, as described above, transactions at the distributed sites 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 the updated granules, shown as command CENTRAL_UPDATE in Fig. 4, any central transactions holding these locks are marked for abort, the data is updated and an acknowledgment is sent back to the distributed site, where the coherence count is decremented as described above. At transaction commit time (command CENTRAL_COMMIT in Fig. 4), 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 blocks updated by the transaction. This is referred to as the authentication phase. The distributed sites (command CENTRAL_COMMIT_1 in Fig. 3) examine the lock status (both concurrency and coherency); if the coherency control status is not null (i.e., there are some in flight asynchronous updates to the data) then an abort message is sent to the central site, causing the central transaction to abort as explained below. If the coherency control field is null, and the locks requested by the central transaction are compatible with the locks currently held at the distributed site, then the locks are also granted to the central transaction, and a positive acknowledgment returned to the central site. If the locks held at the distributed site are incompatible with those requested by the central transaction, the local transactions holding these locks are marked for abort, and the central transaction is granted the locks (for the duration of the commit phase).

If the central site receives a positive acknowledgment from all the involved distributed sites (command COMMIT1 with type ACK in Fig. 4) it sends a commit message to all the involved sites (command CENTRAL_COMMIT_2 in Figs. 3 and 4); if an abort message is received from any of the distributed sites (command COMMIT1 with type ABORT in

```
central_cc_mgr(command,trx_id,gran_list,type)
{ /* central site concurrency and coherency manager actions */
  if(command = = LOCK) /* locking used for conc. control among central trans. */
  { if(compatible) /* compatible with current locks held */
    { grant lock ; return message(GRANTED) }
    else
    { if(deadlock) /* standard check for cycle of waiters at local site only */
      { release all locks held by transaction trx_id and grant locks to compatible waiters
        return message(ABORT) /* abort transaction */
      }
      else
      { enqueue ; return message(LOCK_CONFLICT_WAIT) } /* place on wait queue for granule */
    }
  }
  if(command = = CENTRAL_COMMIT) /* commit of a centrally running class B trans. */
  { if(transaction marked for abort)
    { /* update from a committed local trans. can mark a central trans. for abort */
      release all locks held by transaction trx_id and grant locks to compatible waiters
      return(ABORT)
    }
    else /* begin first phase of commit */
    { build local_site_list /* primary local sites for granules accessed */
      set count of pending responses for transaction trx_id to number sites in local_site_list
      for all local sites in local_site_list
      { send local_cc_mgr(CENTRAL_COMMIT_1,trx_id,gran_list) }
    }
  }
  if(command = = COMMIT1) /* local site response to first phase commit */
  { if(type = = ABORT)
    { mark transaction trx_id for abort }
    decrement count of pending responses for transaction trx_id
    if(all response messages received)
    { /* end of first phase commit */
      if(transaction trx_id marked for abort)
      { for all local sites in local_site_list
        { send local_cc_mgr(CENTRAL_COMMIT_2,trx_id,ABORT) }
      }
      else
      { for all local sites in local_site_list
        { send local_cc_mgr(CENTRAL_COMMIT_2,trx_id,gran_value_list,COMMIT) }
      }
    }
  }
  if(command = = CENTRAL_UPDATE) /* asynchronous update received from local site */
  { for all granules in gran_list
    { if(granule held in incompatible mode)
      { /* abort conflicting central transaction */
        mark holder of incompatible central transactions for abort
      }
    }
  }
  if(command = = COMMIT2) /* responses from local site for second phase of commit */
  { if(all response messages received)
    { write log /* this can be done asynchronously outside CC manager */ }
  }
}
```

Fig. 4. Pseudo-code for central site concurrency–coherency control (balanced protocol).

Fig. 4) the transaction is aborted and re-executed, and the process repeats. When the distributed sites receive the second phase commit message (command CENTRAL_COMMIT_2 in Fig. 3), the database is updated and the locks corresponding to the transaction at the central site are released. (Note that in the rare event that a central transaction conflicts with more than one local transaction at different sites, then there is a small possibility that the central transaction may abort one of the conflicting local transactions, and be aborted itself by another local transaction. While this event can occur, the probability of occurrence is so small that it was never detected in simulations.)

To summarize the above protocol, locking is used at the distributed sites (respectively, central site) for concurrency control among transactions at the same site. The concurrency control among transactions running at different sites is optimistic, and the first transaction that requests a commit at the master site causes an abort of any uncommitted running transactions. Class A transactions running at the master (distributed) site, are marked for abort if a class B transaction requests a commit at the master site and is found to have updated a granule locked by the class A transaction at the master site. A Class B transaction is marked for abort if an update of a granule it has locked is received from a distributed site, and it is aborted at commit time either if it was previously marked for abort or if an update from the distributed site is in transit to the central site. The latter case is detected by keeping tabs of in-flight updates. We note that deadlocks cannot occur among transactions at different sites because an optimistic protocol is used. Infinite aborts do not occur because a transaction is only aborted by a committing transaction (except for the rare case mentioned in the previous paragraph).

Balanced protocol with the central site as master: this protocol is the complement of the above protocol obtained by reversing the roles of the central and local sites. In this protocol, central transactions commit without any communication with the local site, while local transactions communicate with the central site at commit time using analogous protocols to the above.

Locally oriented protocols: the differences between the balanced protocols and the corresponding locally oriented protocols are as follows. In the latter, class A transactions are never aborted (and hence no check for local aborts is made at commit of the class A transaction). During the

authentication phase of class B transactions, if a class A transaction is found to hold the lock in an incompatible mode, the central transaction is aborted (as commented in Fig. 3 for command CENTRAL_COMMIT_1). An aborted central transaction releases all locks and waits for a back off period before restarting. The back off period is to ensure that the class A transaction with which the contention occurred has completed and propagated any updates to the central site. Locks are released at this time so that transactions waiting on other locks held by this transaction are not held up during this back off period [8]. (This is different from the balanced protocol described above where locks were not released at the central site when a class B transaction was aborted. The rationale is that, for the balanced protocol, there is no need for a back off period because the conflicting class A transaction has already committed. Further, all granules accessed by the class B transaction will be in main memory (as argued earlier) and the second run time is very short.) A variation of the locally oriented scheme is also examined here in which locks continue to be retained by aborted transactions during the back off period, rather than releasing the locks as described above. This is referred to as lock retention.

Pessimistic protocol: for completeness, we compare the performance of these schemes with a pessimistic protocol (with local site as master) in which a centrally running transaction explicitly obtains a lock from a local site when the lock request is made (rather than in the optimistic manner as described above).

Note that the protocols can be extended to a third class C of transactions that occasionally require remotely located data. Class C transactions run at local systems and make remote function call requests [11] directly to the remote sites involved. Alternatively, class C transactions can make remote calls to the central site. While such a protocol is a straightforward extension of the above, it is not analyzed in detail here.

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 discussed in this section. The details of the data contention model are presented in the Appendix. Notice that data contention between local and central transactions is manifested as an abort of one of the transactions, 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 [25]. The approach converges very rapidly in a few iterations.

We describe in detail the analysis of the balanced protocol with local site as master. The balanced protocol is chosen not only because it leads to better performance (as we shall see in Section IV), but also because it is the one which is more difficult to analyze. For example, compared with the locally oriented protocol where only local transactions can get aborted, both local and central transactions can get aborted under the balanced protocol. This adds additional complexity to the analysis. 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 arrival rate. A Glossary of all notations used in the analysis is provided above.

3.1 Overview

The complexity of the response time analysis comes from the fact that two extra components are introduced in estimating the response time. One is the waiting time for locks due to lock contentions. This can be estimated from the lock contention probability and the wait time for each contention. However, while the transaction response time depends upon the lock contention probability and wait time for each contention, the contention probability and this wait time also increase with transaction response time. After the contention probability and wait time are analyzed, an iterative approach is employed to capture the feedback effect. The other component is the repeated execution due to transaction abort. The abort probability can be derived from the probability of conflict between local and central transactions similar to the analysis of contention probability. The rerun requirement is then reflected in the queueing model for hardware resource contention through the iteration. Recall that the transaction reaching commit phase first aborts the other transaction. Note that the central and local transactions, and transactions running for the first time and rerun transactions, all have different execution times. Furthermore, there is a "nonuniformity" phenomenon in that a transaction is more likely to have a conflict with a transaction holding more locks. These effects make the analysis of abortion and contention interesting.

Because of the difference in their response times we distinguish among four kinds of transactions: i) newly arrived transactions assigned to run at local sites, with average response time of R_{TL1}; ii) transactions that run at the local site after being aborted at least once (re-run local transactions) with average response time of R_{TL2}; iii) newly arrived transactions assigned to the central site (that run for the first time), with response time R_{TC1}; and iv) transactions that run at the central site after being aborted at least once (re-run transactions) with average response time of R_{TC2}. All locks are assumed to be held until the end of the transaction. For 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 set-up phase (during which no locks are held). A transaction that is re-run after an abort is modeled to find all data referenced in its main memory. Further, locks at either the local (respectively, central) site are not released after an abort. The difference between R_{TL1} and R_{TL2} (or R_{TC1} and R_{TC2}) is the I/O time, processing time, and wait time for locks.

In Sections 3.2 and 3.3, we derive an expression for the local (central) transaction response time in terms of contention

and abort probabilities. In the Appendix, the wait time for each lock contention is derived through a transaction flow diagram to capture the nonuniformity phenomenon. Also in the Appendix, 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 local (respectively central) transactions and between local and central transactions are derived. Conflicts among transactions running at the same site are mapped into lock waits, while conflicts between local and central transactions are mapped into aborts of either the local or the central transaction.

3.2 Response Time for Local Transactions

In order to evaluate the local transaction response times, transaction arrivals are modeled by a Poisson process. Assuming that aborts are independent events with identical abort probability for all aborts after the first, and that rerun times are i.i.d., the average total local transaction response time R_{TL} is expressed as

$$R_{TL} = R_{TL1} + R_{TL2}\frac{p_{AL}}{1-\tilde{p}_{AL}}$$

where R_{TL1} is the component of the response time for the first run of a local transaction, R_{TL2} is the response time component for a re-run of a local transaction that was aborted, p_{AL} is the probability of the first local abort, and $\tilde{p}_{AL}$ is the probability of subsequent local transaction aborts. In turn, R_{TL1} can be expressed in terms of its components as follows:

$$R_{TL1} = R_{CPUL1} + R_{IOL1} + R_{CONTL1}.$$

Each component will be described separately.

R_{CPUL1} 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 (10 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 (15 per transaction) 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. 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. 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 are evaluated by modeling the CPU as an M/M/1 queue. Then R_{CPUL1} is expressed as

$$R_{CPUL1} = R^{INPL}_{CPUL1} + R^{PROC}_{CPUL1}$$

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 transaction/application processing.

R_{IOL1} 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,

$$R_{IOL1} = t_{IO}n_{IO}$$

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} (5 per transaction) 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. The database I/O's per transaction n_{IODB} (11 per transaction) 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. Then

$$R_{IOL1} = (n_{IOPL} + n_{IODB})t_{IO} = R_{IOPL} + R_{IODB}.$$

R_{CONTL1} is the time spent in contention wait for a lock that is held by another transaction, and is derived in the Appendix, as is the derivation of the probability of transaction abort. The remaining term to evaluate in R_{TL} is R_{TL2}. Recall that in the protocol, local locks are retained even if the transaction is aborted. We assume that all the data for rerun transactions are available in local buffers, and consequently there are no I/O's for the rerun transaction. Thus, $R_{TL2} = R^{PROC}_{CPUL2} + T_{Backoff}$, where R^{PROC}_{CPUL2} is similar to R^{PROC}_{CPUL1} excluding the overheads for locking and I/O, and $T_{Backoff}$ is the time the transaction waits before restarting after an abort.

3.3 Response Time for Central Transactions

The total response time of a transaction running at the central site is expressed as,

$$R_{TC} = R^{SHIP}_{CPUL} + R_{COMDLY} + R_{TC1} + R_{TC2}p_{AC} + R_{TC3}\frac{p_{AC}\tilde{p}_{AC}}{1-\tilde{p}_{AC}} + R_{COMMIT}.$$

R^{SHIP}_{CPUL} is the component of 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. R_{TC1} covers the time to run a new transaction at the central site, and the fourth and fifth terms account for the average re-run time. R_{COMMIT} includes the communications delays to and from the central site for the authentication and second commit phase, the CPU time at the central site and local

sites for communication, authentication and commit phase overhead, and the I/O time and overhead for updating the local database during the second commit phase. At the central site 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 local site involved in the commit there is communications overhead, an overhead for authentication (2K instructions), and an overhead to update the local database. p_{AC} is the probability of the first abort of a central transaction, and $\tilde{p}_{AC}$ is the probability of the second and subsequent aborts of central transactions. Analogous to the development of the response time expressions for the local transaction we write

$$R_{TC1} = R_{CPUC1} + R_{IOPL} + R_{IODB} + R_{CONTC}.$$

The component R_{CPUC1} equals $R_{CPUC1}^{INPL} + R_{CPUC1}^{PROC}$ to execute the first run of a transaction at the central site, and is similar to that for the local transaction described above. As before, an M/M/1 model is used to estimate R_{CPUC1}^{PROC} as well as R_{CPUL}^{SHIP} and R_{COMMIT}. R_{IOPL} and R_{IODB} at the central site are the same as those at the local site. R_{CONTC} is the average wait time due to contentions with the other central transactions and is estimated in the Appendix. The remaining terms to evaluate in the above expression are R_{TC2} and R_{TC3}. Recall that in the protocol, central locks are retained even if the transaction is aborted. We assume that all the data for rerun transactions are available in central buffers, and consequently there are no I/O's for the rerun transaction. Thus, $R_{TC2} = R_{CPUC2}^{PROC} + P_{GL1}R_{COHER}$, and $R_{TC3} = R_{CPUC2}^{PROC} + P_{GL2}R_{COHER}$, where R_{CPUC2}^{PROC} is the same as R_{CPUC1}^{PROC} excluding the overheads for locking and I/O, and R_{COHER} includes communications delays to and from the central site, and CPU time at the central site and local sites for communication and authentication phase overhead. P_{GL1} (respectively, P_{GL2}) is the probability that the first (respectively, subsequent) abort is detected after communication with the local sites during the authentication phase, and the methodology to derive these probabilities is given in the Appendix.

IV. Results and Discussion

The methodology of Section III has been used and validated extensively in developing approximate analytical models of centralized systems [25], [28], [29] and geographically distributed systems [7], [8]. We first present some results of the analysis of the preceding section with simulation estimates. Following this, we focus on further projections from the approximate analytical model. Further validation was also carried out for several points as projected by the analytical model in Section 4.2.

4.1. Simulation

Both the balanced and locally oriented 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 ms communications delay; the central CPU has 10 MIPS while each local site has 1 MIPS. Transaction 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 instruction pathlengths used in both the analytical and simulation models are summarized in Table I. In the simulation, a global lock space of 32K elements ($LSPACE$ in the Appendix) 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. The simulation maintains a lock table and explicitly simulates lock contention, and waits for locked entities, queueing and processing at the CPU, communication delay and overhead, I/O waits, aborts of central and/or local transactions, and commit processing. The CPU service times correspond to the time to execute the specific instruction pathlengths given in Section 3.2 (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. To reduce simulation time, 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. 5 shows the probability of the first abort versus total transaction rate for central transactions, for both protocols. The estimates for the simulation and analysis are very close. Fig. 6 shows the average local and central transaction response times versus total transaction rate, for both protocols. The difference between analysis and simulation is within 5% for processor utilizations of less than about 60%, and is within 10% for higher utilizations. Note the simulation is rather time consuming even for this modest configuration, taking about 10 to 20 min of CPU time on an IBM 3090 for each point simulated. We present the comparison study in Section 4.2 based on the approximate analysis for configurations with a larger number of nodes and a faster CPU where it would be too time consuming to simulate. (Further validation was also carried out for several points as projected by the analytical model and the simulations took a couple of hours of CPU time for each point.)

4.2. Comparison of Protocol Performance

We now present some further performance projections from the model. We illustrate the trade-offs between the approaches for the case of 16 geographically distributed sites. The results are similar for other numbers of distributed sites. A global lock space of 50K elements (*LSPACE* of Appendix I) is used in the following charts. This is roughly the effective database size corresponding to the measured lock contention probability in trace driven simulations on which the transaction characteristics given in Section III are based [25]. The other parameters and overheads are the same as given in Section III. We examine the effect of changing the mix of transaction types, varying the communication delay and the transaction rate.

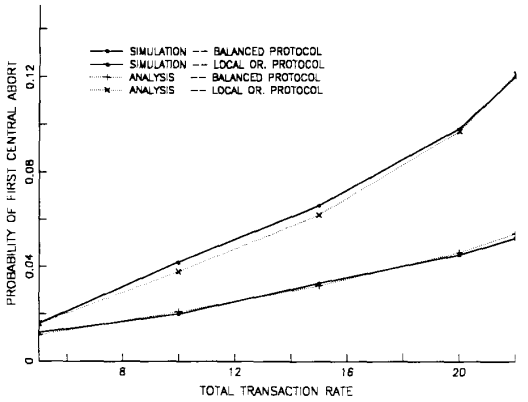

Fig. 5. Abort probability versus transaction rate.

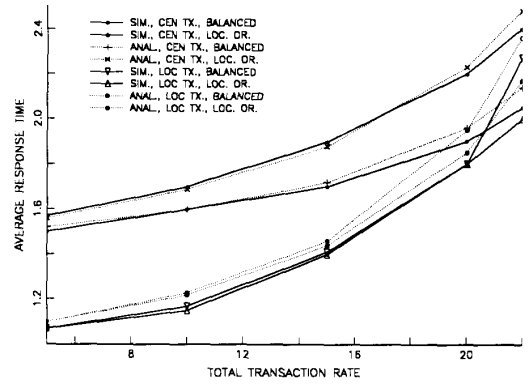

Fig. 6. Response time versus transaction rate.

In [13] it is shown that a distributed system with no data replication requires very strong locality of data reference to achieve good performance. The intent of the hybrid system structure is to relax the locality requirement so that the system can work well with modest locality. Fig. 7 shows that the improvement in robustness to locality depends upon the protocol chosen. The average response time versus fraction of local transactions (i.e., fraction of class A transactions) is presented for the five protocols: balanced protocols with local sites and central site as masters, respectively, locally oriented protocols with and without lock retention, and the pessimistic protocols. The graph is for a total of 80 MIPS. The MIPS at the central and local sites are adjusted so as to minimize the average response time, while constraining the total to be 80 MIPS. A communication delay of 200 ms is examined, where the communication delay is the difference in the times when a message is sent out from a system and when it is received by a destination system. The back off time before restart of an aborted local transaction $T_{Backoff}$ is set to twice the communication delay so as to ensure that the conflicting central transaction has completed the commit processing. For comparison, the characteristic of a fully distributed database system without replication is also indicated in the figure. The model is based on the analysis in [13]. In the fully distributed system, for all access to non-local data, a remote function call is made to the system that has the data. Hence, such a system is very sensitive to the fraction of local transactions, as the chart indicates. The hybrid system using the pessimistic protocol is slightly worse than the fully distributed system, because a remote call is made for every lock request, rather than for every function call. For all the other protocols considered, the hybrid system is much less sensitive to the fraction of local transactions than the fully distributed system or the hybrid system with the pessimistic protocol. This is primarily because of a much lower number of average communications per transaction due in part to the data replication at the central site, and to the optimistic protocol for intersite concurrency control. The balanced protocol with central site master is very insensitive to the fraction of local transactions as it does not take full advantage of locality. It has the best performance when the fraction of local transactions is small and also the worst performance when the fraction of local transactions is high. The reason for the insensitivity of this protocol to the fraction of local transactions is the low overhead for commit processing: Local transactions have a (single phase) commit process to the central site, and central transactions commit without any communications. By comparison, the protocols with the local sites as master require that central transactions have a two-phase commit process with all of the local sites that are masters for the granules accessed. Therefore, for the remaining protocols with local site master, as the fraction of local transactions decreases, the average response time increases; however, the response time of the balanced protocol with local site master increases at a slower rate than the others. The reason is that the probability of local central conflicts increases with the fraction of central transactions in this range, thus increasing the probability of aborts and hence, the response time. In the locally oriented protocols with or without lock retentions, all conflicts result in central transaction aborts leading to a much higher central transaction response time and hence, to a larger average response time. Thus, the balanced protocols are significantly more stable than the other protocols with respect to the fraction of local transactions. Comparing the two locally oriented protocols, the case with lock retention by a central transaction during the back off time performs worse than the case when locks are released. This is due to an increase in the lock contention probability, and consequent increase in the lock wait time at the central site. Note that this back off time is necessitated by the communication delay and there is negligible difference between these two cases for very small communication delay, as described later.

The effects described above are further illustrated in Fig. 8, which shows the average response time versus transaction rate for a 75% fraction of local transactions. A total of 160 MIPS and a communication delay of 200 ms are assumed. As above, the MIPS at the local and central sites are adjusted to obtain the minimum response time. The balanced protocol with local site master always has a smaller response time than the locally oriented protocols with or without lock retention and the pessimistic protocol; furthermore, it can support a larger transaction rate. The balanced protocol with central site master does very well when the transaction rate is high. The

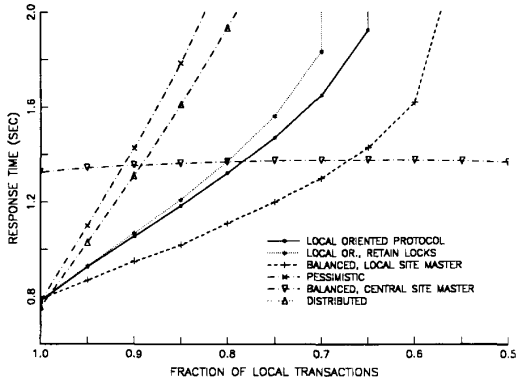

Fig. 7. Response time versus locality.

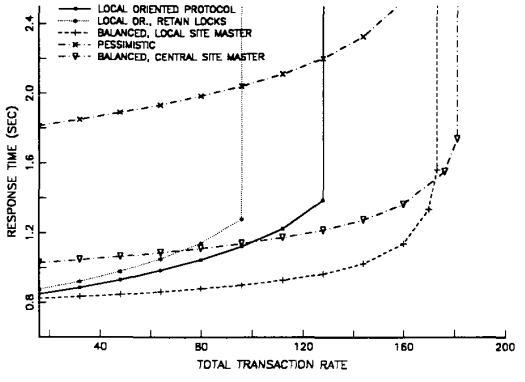

Fig. 8. Response time versus transaction rate.

conflicts between local and central transactions increases with transaction rate leading to a similar effect to that described above. We chose a total of 160 MIPS to illustrate this effect, because with fewer MIPS, the limitation in transaction rate is due primarily to the limited MIPS rather than due to aborts. It is at higher conflict levels (e.g., higher transaction rates or a larger fraction of central transactions) that the difference in the protocols is accentuated.

In Fig. 9 we examine the minimum MIPS required to support a total transaction rate of 80 transactions per second, while providing an average response time of no more than 1.5 s for a communication delay of 200 ms. An iteration on the total MIPS at the central and local sites is used to find the minimum MIPS satisfying the response time constraint. Again, the distribution of MIPS between the central and local sites is chosen so as to minimize the average response time obtained. The figure indicates that the pessimistic protocol is extremely sensitive to the fraction of local transactions. The locally oriented protocols with and without lock retention, and the balanced protocols are much less sensitive to this factor, with the latter protocols being superior in this respect. Again it is the larger number of average communications per transaction that makes the pessimistic protocol worse than the other protocols that use optimistic intersite concurrency control. For the locally

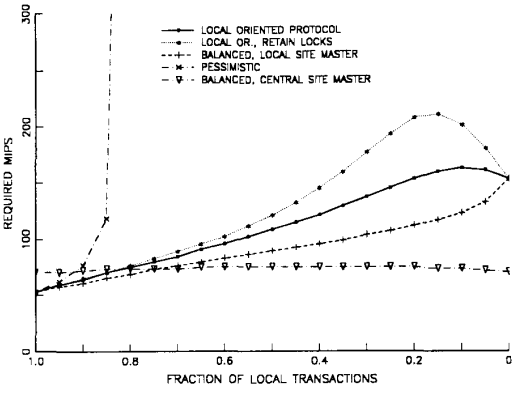

Fig. 9. MIPS required versus locality—worst case.

oriented protocols, the MIPS required has a maximum for an intermediate fraction of local transactions between 0 and 1. The reason is that intersite conflicts are larger in this intermediate region giving rise to a larger transaction abort probability, that leads to an additional processing requirement. The balanced protocol with local site master does not exhibit this phenomenon because the overall probability of aborts is smaller (as described earlier) and because the additional commit processing overhead for a smaller fraction of local transactions dominates over the effect of the abort probability. (It is possible for the balanced protocols to exhibit a similar "bell-shaped" characteristic for parameter values that give rise to a higher abort probability.) For the balanced protocol with central site master the response time actually decreases for a smaller fraction of local transactions. The reason is that this protocol does not make much use of locality, and requires that a local transaction communicate with the (master) central site at commit time. Instead of this communication the transaction could be shipped to the central site, eliminating contention among transactions running at different sites. There is a cross-over in the characteristics of the balanced protocols with central site master and local site master. This is because, as explained earlier, the protocols with central site master do not take advantage of locality, and because of their relative insensitivity to the fraction of local transactions.

In Fig. 9 we have assumed that for central transactions the number of distributed sites in commit processing is the number of database calls (i.e., all remote database calls are to different systems). This is referred to as the worst case scenario. In Fig. 10, we present the best case when all remote database accesses are to the same site. The cross-over point between balanced protocols with local site master and central site master shifts to a lower fraction of local transactions, i.e., the protocol with local site master looks more favorable under the best scenario. This is because the number of communications during the commit process decreases for this protocol, but is unchanged for the case with the central site as master (only one master site to communicate with). In both cases, the balanced protocol with local site master does well with strong or modest locality, while the protocol with central site master does worse when there is strong locality but performs better at modest to low locality.

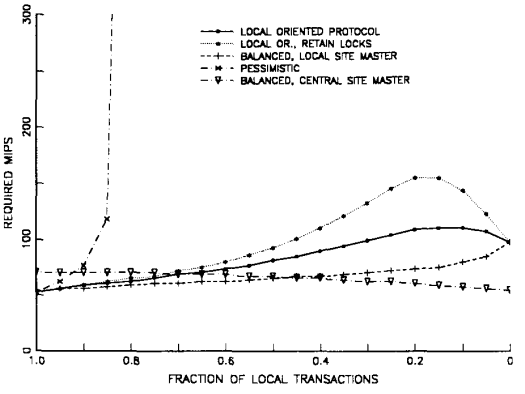

Fig. 10. MIPS required versus locality—best case.

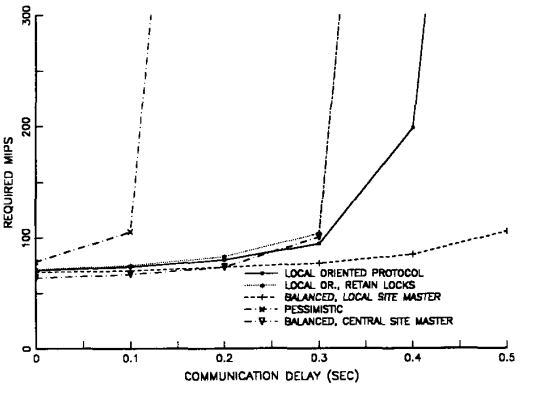

Fig. 12. MIPS required versus communication delay.

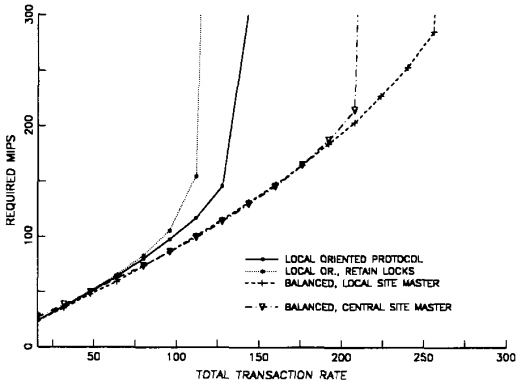

Fig. 11. MIPS required versus total transaction rate.

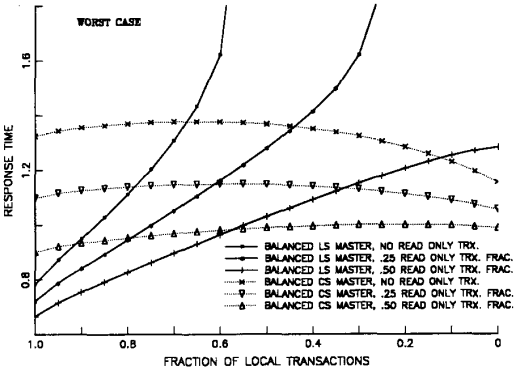

Fig. 13. Effect of read only transactions—worst case.

Fig. 11 shows the MIPS required to support a varying transaction rate, with a 1.5-s response time bound, assuming 75% of local transactions, and a communication delay of 200 ms. As in Fig. 8, the conflicts increase with transaction rate, and the balanced protocols require fewer MIPS to meet the constraints. Fig. 12 shows the MIPS required to support a varying communication delay, with a 1.5-s response time bound, assuming 75% of local transactions and a total transaction rate of 80 transactions per second. The balanced protocol with local site master is much less sensitive to the communication delay as compared to the other protocols. This again is because this protocol provides a better balance of aborts caused by intersite conflicts.

Finally, we consider the effect of relaxing the consistency requirement. The coherency control requirement is relaxed for the read-only transactions that are shipped to the central site to allow them to read data that is concurrently updated by a transaction running at a local system (or vice versa for the protocol with the central site as master). That is, read-only transactions shipped to the central site commit without any communication with the local sites. The balanced protocols with local site master and central site master are compared under different fractions of read only transactions in Figs. 13 and 14 for the worst case and best case scenarios, respectively. Again, the worst case assumes that all remote calls are directed to different sites, whereas the best case assume that all remote calls are to the same site. A total of 80 MIPS and a communication delay of 200 ms are used with a total transaction rate of 80 transactions per second. As in Fig. 7, there is a cross-over in the characteristics for the two protocols. That is, below the cross-over fraction of local transactions, the balanced protocol with central site master has a smaller response time. For the worst case scenario in Fig. 13, the cross-over point for the fraction of local transactions drops from 0.67 to 0.55 (making the protocol with local site master more attractive) when the read-only fraction increases from 0 to 50%. For the best case scenario in Fig. 14, the cross-over points are more or less unchanged at about 45% local transactions. Notice that for both protocols, the response time becomes less sensitive to the fraction of local transactions as the fraction of read-only transactions increases. Since the environments we are considering exhibit regional locality, the balanced protocol with local site master would be the protocol of choice.

V. Conclusions

In this paper, we examined the issue of how to enhance the robustness of distributed transaction processing systems so that performance would be adequate even with modest locality. This is in contrast to fully distributed systems with

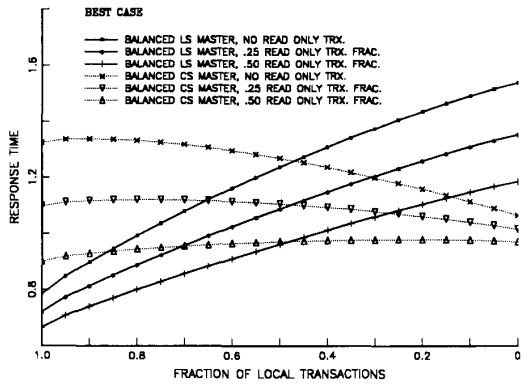

Fig. 14. Effect of read only transactions—best case.

no replication that are extremely sensitive to the fraction of nonlocal data reference. A hybrid distributed and centralized system structure is studied and various protocols to maintain concurrency and coherency controls for transactions running at the local and central systems are proposed and analyzed. An approximate analysis method is developed to estimate the system performance. The analysis captures the distributed structure of the hybrid system and the intricacies of the different protocols. The analysis decomposes the effect of hardware resource contention from data contention and uses an iteration to model the interaction between the two effects. A straightforward but accurate approach is used for estimating the lock contention and transaction abort probabilities, based on the holding time of data items and the request rate. We identify the critical protocol design factors affecting the performance. These include 1) the approach for the intersite concurrency control protocol, optimistic versus pessimistic, 2) the resolution of abort due to intersite conflict, balanced versus locally oriented, 3) the master site of the dual copies, local site versus central site. It is found that by judiciously designing the protocol, the robustness of the system can be greatly enhanced. A class of protocols using an optimistic protocol for intersystem control shows significantly lower sensitivity to the number of nonlocal data references. These protocols allow locally running transactions to commit without any communication with the central site. Among this class of protocols, the balanced protocols attempt to further split aborts due to intersystem data conflict evenly between aborts of local and of central transactions. This allows a larger overall transaction rate to be supported. These protocols also show the least sensitivity to the fraction of nonlocal transactions. The balanced protocol can operate either with local site or central site as master of concurrency and coherency control. For the environments we consider, with some locality of reference, the balanced protocol with local site master is the protocol of choice. This hybrid system structure also lends itself to load balancing, which is a topic of further research.

VI. Appendix

In Section III, the local and central transaction response times were derived in terms of transaction conflict and abort probabilities, and contention wait time for the balanced protocol with local site as master. In this Appendix, we derive the transaction wait times due to contention, and the conflict and abort probabilities for this case. The contention wait time estimation is an extension of that in [25]. The abort probabilities are based on conflict probabilities between transactions running at different sites.

A.1. Contention Wait Time Estimation

R_{CONTL1} is the time spent in contention wait by a local transaction for a lock that is held by another transaction. The contention wait is estimated as,

$$R_{CONTL1} = R_{CONTL} + R_{CONTC}^{COHER}$$

where R_{CONTL} is the time spent in contention wait for a lock that is held by another local transaction and R_{CONTC}^{COHER} is the time spent in contention wait for a lock that is held by a central transaction during authentication phase. We use the following approach to evaluate R_{CONTL}.

$$R_{CONTL} = N_L(P_{LOC.LOC1}\bar{W}_{L1} + P_{LOC.LOC2}\bar{W}_{L2})$$

where N_L is the number of locks per transaction, $P_{LOC.LOC1}$ (respectively, $P_{LOC.LOC2}$) is the probability of contention on a lock request with a new local transaction (respectively, rerun local transaction), and $\bar{W}_{L1}$ (respectively, $\bar{W}_{L2}$) is the average time a local transaction waits if it contends with a new local transaction (respectively, rerun local transaction) for a lock. The evaluation of $P_{LOC.LOC1}$ and $P_{LOC.LOC2}$ is given in Section A.2 of this Appendix. The wait time is the residual time for the transaction holding the lock to reach the end of its current run, plus the time for any rerun needed (since locks are not released when a transaction is aborted). This residual time is one third of the run time if locks are acquired uniformly through the run, or one half if all locks are acquired at the beginning of the run [25]. The wait time $\bar{W}_{L1}$ is estimated as $\bar{W}_{L1} = \beta_l/F + \gamma_l$, where $\beta_l = R_{CPUL1}^{PROC} + R_{IODB} + R_{CONTL1}$ is the period during which locks are acquired during the first run of a transaction, $\gamma_l = R_{TL2} \times p_{AL}/(1 - \tilde{p}_{AL})$ is the rerun period during which no further locks are acquired, and F a factor that depends on the manner in which locks are acquired during the first execution of the transaction. Assuming that locks are acquired uniformly during the period β_l, the factor F is estimated as 3 [25]. The wait time $\bar{W}_{L2}$ is estimated as $\bar{W}_{L2} = R_{TL2}/2 + \gamma_l$, by similar reasoning to that for $\bar{W}_{L1}$, except that local locks are not released after the first run of a local transaction and hence a factor of 2 rather than 3 is used. Similarly, R_{CONTC}^{COHER} is estimated as,

$$R_{CONTC}^{COHER} = N_L P_{LOC.CEN}^{COHER} \frac{R_{HOLD}}{2}$$

where $P_{LOC.CEN}^{COHER}$ is the probability of contention on a lock request with another transaction running on the central site that is in the authentication phase. The manner in which $P_{LOC.CEN}^{COHER}$ is estimated is described later. R_{HOLD} is the amount of time a central transaction holds locks at the local site during its authentication phase. It is the time spent in the CPU for authentication estimated from an M/M/1 model plus the communication delay. Since all locks are obtained at the

$$P_{CEN.CEN2} = \frac{\lambda(1-p_{LOC})N_{DS}N_L\gamma_c}{LSPACE}$$

$$P_{LOC.LOC1} = \frac{\lambda p_{LOC}N_L(\beta_l/2)}{LSPACE/N_{DS}}$$

$$P_{LOC.LOC2} = \frac{\lambda p_{LOC}N_L\gamma_l}{LSPACE/ < N_{DS}}$$

$$P_{LOC.CEN}^{COHER} = \frac{\lambda(1-p_{LOC})N_L(R_{COHER}(1+p_{AC}\times p_{GL1}+(p_{AC}\times\tilde{p}_{AC}\times p_{GL2})/(1-\tilde{p}_{AC}))+R_{COMMIT})}{LSPACE/N_{DS}}.$$

beginning of the authentication phase, the average wait time for such a contention is estimated as $R_{HOLD}/2$ in the above equation.

We evaluate R_{CONTC} in an analogous manner to R_{CONTL} as

$$R_{CONTC} = N_L(P_{CEN.CEN1}\bar{W}_{C1} + P_{CEN.CEN2}\bar{W}_{C2}).$$

The terms in this equation are analogous to those for evaluating R_{CONTL}. $\bar{W}_{C1}$ and $\bar{W}_{C2}$ can be evaluated in a similar manner to that above. Define $\beta_c = R_{CPUC1}^{PROC} + R_{IODB} + R_{CONTC}$, $\gamma_c = \gamma_1 + \gamma_2$, $\gamma_1 = p_{AC}(\delta_1 + \delta_2\tilde{p}_{AC}/(1-\tilde{p}_{AC}))$, $\delta_1 = R_{TC2}$, $\delta_2 = R_{TC3}$, $\gamma_2 = R_{COHER} + R_{COMMIT}$. Note that β_c is the period during which locks are acquired by a newly arrived transaction at the central site, and γ_c is the period during which no further locks are acquired. The period γ_c is comprised of a period γ_1 for reruns of transactions, and a period γ_2 for coherence check and commit processing. Then, $\bar{W}_{C1}$ can be approximated as, $\bar{W}_{C1} = \beta_c/3 + \gamma_c$, in a similar manner to $\bar{W}_{L1}$, and

$$\bar{W}_{C2} = \frac{\gamma_1}{\gamma_c}\left\{\frac{p_{AC}\delta_1}{\gamma_1}\left(\frac{\delta_1}{2} + \frac{\tilde{p}_{AC}}{1-\tilde{p}_{AC}}\delta_2 + \gamma_2\right) + \frac{\delta_2 p_{AC}\tilde{p}_{AC}}{\gamma_1(1-\tilde{p}_{AC})}\left(\frac{\delta_2}{2} + \frac{\tilde{p}_{AC}}{1-\tilde{p}_{AC}}\delta_2 + \gamma_2\right)\right\} + \frac{\gamma_2}{\gamma_c}\frac{\gamma_2}{2}$$

In this equation, γ_1/γ_c is the probability of contention with a central transaction in the rerun phase, while the term γ_2/γ_c is the probability of contention with a central transaction in the coherence check or commit phase. For the former case, the transaction in question may either be in the second run (probability of $(p_{AC}\delta_1/\gamma_1)$ or in a subsequent run (probability of $(\delta_2 P_{AC}\tilde{P}_{AC}/\gamma_1(1-\tilde{p}_{AC}))$), and for each of these cases, a wait time is estimated. Note that if $\delta_1 = \delta_2$ and $\gamma_2 = 0$, then the above expression for $\bar{W}_{C2}$ becomes similar to that for $\bar{W}_{L2}$.

A.2 Contention and Abort Probabilities

We now derive the lock contention and 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 local (respectively, central) transactions are manifested as lock contention wait for the element to be unlocked by the local (respectively, central) transaction holding the lock. Conflicts between local and central transactions manifest themselves as aborts of either the local or central transaction.

Each local system has a local database and is subject to a transaction rate λ. Of these transactions, a fraction p_{LOC} execute locally. 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, thus offering a transaction arrival load of $\lambda(1-p_{LOC}) \times N_{DS}$ at the central site, where N_{DS} is the number of local systems. As defined in the previous section, $P_{CEN.CEN1}$ (respectively, $P_{CEN.CEN2}$) is the probability of a contention on a lock request by a central transaction with a new (respectively, rerun) central transaction. We estimate,

$$P_{CEN.CEN1} = \frac{\lambda(1-p_{LOC})N_{DS}N_L(\beta_c/2)}{LSPACE}$$

In this equation, the numerator is the average number of locks held by first run transactions at the central site and is obtained using Little's formula; assuming uniform access of granules over the database, dividing this quantity by the number of granules of the database $LSPACE$ gives the probability that a lock is already held by a first run central transaction when another central transaction makes a lock request. Similarly we write four equations (see top of page). In each of these equations, the numerator estimates the average number of locks held by the type of transaction involved, and the denominator is the space over which the lock requests are made. Notice that in estimating the contention among local transactions, *LSPACE* is divided by the number of distributed systems, since it is assumed that local transaction makes lock requests only in their own partition.

We now estimate the transaction abort probabilities. The key observation is that local transactions are only aborted by committing central transactions, and vice versa, and that the vulnerability period for being aborted depends on a conflict with a committing transaction between accessing the data item and the commit point. Simulations show that the following approximate method for estimating abort probabilities can be applied to optimistic protocols in general and is very accurate. The probability of abort of a new local transaction P_{AL} is estimated

as

$$P_{AL} = \frac{\lambda(1 - P_{LOC})(\beta_l/2)N_L^2}{LSPACE/N_{DS}}$$

In this equation, $\lambda(1 - P_{LOC})N_L$ is the rate at which committing central transaction reference local data items at a particular site. $\beta_l/2$ is the average lock holding time of local transactions. Hence, $\lambda(1 - P_{LOC})N_L(\beta_l/2)$ is the average number of local data items referenced by committing central transactions during the average lock holding time of a local transaction. Dividing this by $LSPACE/N_{DS}$ estimates the probability that a local transaction accesses a data item that is referenced by a committing central transaction over its first run, causing it to abort. Assuming that at most one such contention can occur per local transaction, the transaction abort probability 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 raising this to the power of N_L to estimate the probability of no abort for the transaction. The difference between such an estimate and the above was found to be negligible except at very high abort probabilities. Similarly the other transaction abort probabilities are estimated as

$$\tilde{P}_{AL} = \frac{\lambda(1 - P_{LOC})\gamma_l N_L^2}{LSPACE/N_{DS}}$$

$$P_{AC} = \frac{\lambda N_{DS} P_{LOC}((\beta_c/2) + 2 \times COMDEL)N_L^2}{LSPACE}$$

$$\tilde{P}_{AC} = \frac{\lambda N_{DS} P_{LOC}(\gamma_c + 2 \times COMDEL)N_L^2}{LSPACE}.$$

The principal difference in estimating the abort probability of central transactions is addition of twice the communications delay $COMDEL$ from the local to the central site to the average central transaction holding time in estimating the locks invalidated due to local transaction commit. One factor of $COMDEL$ arises because (in flight) asynchronous updates of local transactions arrive at the central site while the central transaction is in transit to the local sites for coherence check, and the second $COMDEL$ factor arises because local transactions may commit and begin asynchronous communications to the central site during this same interval. Recall from Section 3.3 that the factor P_{GL1} (respectively, P_{GL2}) is defined as the probability that the first (respectively, subsequent) abort is detected after communication with the local sites during the authentication phase. In the above equations estimating P_{AC} and $\tilde{P}_{AC}$, the term involving twice the communications delay is due to aborts detected after communications with the central site, and the term with $\beta_c/2$ (respectively, γ_c) is due to aborts without any communications to the central site. Hence, we estimate $P_{GL1} = (2 \times COMDEL)/(\beta_c/2 + 2 \times COMDEL)$ $P_{GL2} = (2 \times COMDEL)/(\gamma_c + 2 \times COMDEL)$.

TABLE I
MODEL PARAMETERS

Pre-processing overhead (to determine trans. type)	20 K instructions
Communications overhead	10 K at each of sending and receiving sites
Initial Processing (no locks held)	150 K instructions
Initial IO (no locks held)	5 / transaction
Database (DB) calls	10 / transaction
Processing / DB call	25 K instructions
Lock requests	15 / transaction
Processing / lock request	1 K instructions
Database IO's	11 / transaction
IO overhead	3 K instructions
Average disk IO time	35 ms
Two phase commit overhead coordinator Local sites	2 K + 3 K × no. systems + comm. overhead per phase 2 K + comm. overhead per phase
Communications delay	0 to 0.5 s
Backoff time for aborted trans.	2 × Comm. delay

GLOSSARY OF NOTATION

λ	Transaction arrival rate at each distributed site
N_{DS}	Number of distributed systems
$LSPACE$	Number of granules in database
$COMDEL$	Average communications delay from local to central site
P_{LOC}	Fraction of class A (local) transactions
R_{TL}	Average Local Transaction response time
R_{TL1}	Average first run time for a local transaction
R_{TL2}	Average run time for a rerun local transaction
R_{CPUL1}	Average CPU service and queueing time in R_{TL1}.
R_{CPUL1}^{INPL}	Average initial processing time in R_{CPUL1}.
R_{CPUL1}^{PROC}	Average transaction/application processing time in R_{CPUL1}.
R_{IOL1}	Average I/O time in R_{TL1}.
t_{IO}	Average time per I/O
n_{IO}	Average number of I/Os per transaction
n_{IOPL}	Average initial I/Os per transaction
n_{IODB}	Average database I/Os per transaction
R_{IOPL}	Average time for n_{IOPL} I/Os
R_{IODB}	Average time for n_{IODB} I/Os
R_{CONTL1}	Average time in lock contention wait in R_{TL1}.
R_{CPUL2}^{PROC}	Average CPU proc. and queueing time in R_{TL2}.
P_{AL}	Probability of first abort of a local transaction
$\tilde{P}_{AL}$	Probability of abort of rerun a local transaction
R_TC	Average shipped (central) transaction response time
R_{CPUL}^{SHIP}	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

R_{TC1}	Average first run time for a central transaction
R_{TC2}	Average second run time for a central transaction
R_{TC3}	Average third and subsequent run time for a central transaction
R_{COMMIT}	Av. Comm. and proc. time for central trans. commit operation
P_{AC}	Probability of first abort of a central transaction
$\tilde{P}_{AC}$	Probability of abort of a rerun central transaction
R_{CPUC1}	Average central transaction first run execution time
R_{CONTC}	Average time in lock contention wait for central trans.
R_{CPUC1}^{INPL}	Initial processing time in R_{CPUC1}.
R_{CPUC1}^{PROC}	Trans./application processing time in R_{CPUC1}.
R_{CPUC2}^{PROC}	Trans./application processing time in R_{TC2}.
P_{GL1}	Prob. that first abort is detected after comm. to local sites
P_{GL2}	Prob. that subsequent abort is detected after comm. to local sites
R_{CONTL}	Portion of R_{CONTL1} for contention with local trans.
R_{CONTC}^{COHER}	Portion of R_{CONTL1} for contention with central trans. in its authentication phase
N_L	Number of locks per transaction
$P_{LOC.LOC1}$	Prob. of local trans. cont. with new local trans.
$P_{LOC.LOC2}$	Prob. of local trans. cont. with rerun local trans.
$\bar{W}_{L1}$	Average wait time associated with $P_{LOC.LOC1}$.
$\bar{W}_{L2}$	Average wait time associated with $P_{LOC.LOC2}$.
β_l	Period during which locks are acquired for local trans.
γ_l	Period during which no further locks are acquired for local trans.
P_{CONTC}^{COHER}	Prob. of contention of a local trans. with a central trans. in its authentication phase
R_{HOLD}	Average time lock held at local site by central trans. in its authentication phase
$P_{CEN.CEN1}$	Prob. of central trans. contention with new central trans.
$P_{CEN.CEN2}$	Prob. of central trans. contention with rerun central trans.
$\bar{W}_{C1}$	Average wait time associated with $P_{CEN.CEN1}$.
$\bar{W}_{C2}$	Average wait time associated with $P_{CEN.CEN2}$.
β_c	Period during which locks are acquired for central trans.
γ_c	Period during which no further locks are acquired for central trans.

ACKNOWLEDGMENT

The authors would like to thank Alberto Avritzer for his contribution in helping develop the simulation program for the hybrid system, Bala Iyer for his work in the early stages of this study, and Asit Dan for his suggestions that resulted in simplification of the analysis of the protocols.

REFERENCES

[1] R.K. Agrawal *et al.*, "Concurrency control performance modeling: Alternatives and implications," *ACM Trans. Database Syst.*, vol. 12, pp. 609–654, Dec. 1987.

[2] R. Balter *et al.*, "Why control of concurrency level in distributed systems is more fundamental than deadlock management," in *Proc. 1st ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing*, pp. 183–193, Aug. 1982.

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

[4] M.J. Carey and M. Livny, "Conflict detection trade-offs for replicated data," *ACM Trans.Database Syst.*, vol. 16, pp. 703–746, Dec. 1991.

[5] W. Cellary *et al.*, *Concurrency Control in Distributed Database Systems*. North-Holland, 1988.

[6] A. Chesnais *et al.*, "On the modeling of parallel access to shared data," *Comm. ACM*, vol. 26, pp. 196–202, Mar. 1983.

[7] B. Ciciani *et al.*, "Analysis of replication in distributed database systems," *IEEE Trans. Knowledge and Data Eng.*, vol. 2, pp. 247–261, 1990.

[8] B. Ciciani *et al.*, "A hybrid distributed centralized system structure for transaction processing," *IEEE Trans. Software Eng.*, vol. 16, pp. 791–806, 1990.

[9] B. Ciciani *et al.*, "Dynamic and static load sharing in hybrid distributed-centralized database systems," *Computer Systems Science and Engineering*, vol. 7, no. 1, Jan. 1992, pp. 25–41.

[10] E. G. Coffman *et al.*, "Optimization of the number of copies in a distributed system," *IEEE Trans. Software Eng.*, vol. SE-7, pp. 78–84, Jan. 1981.

[11] D.W. Cornell *et al.*,"On multisystem coupling through function request shipping," *IEEE Trans. Software Eng.*, vol. SE-12, pp. 1006–1017, Oct. 1986.

[12] A. Dan *et al.*, "The effect of skewed data access on buffer hits and data contention in a data sharing environment," in *Proc. 16th Int. Conf. Very Large Data Bases*, pp. 419–431, Aug. 1990.

[13] D.M. Dias *et al.*,"On centralized versus geographically distributed database systems," in *Proc. 7th Int. Conf. Distributed Computing Systems*, pp. 64–71, Sept. 1987.

[14] B.I. Galler, "Concurrency control performance issues," Ph.D. dissertation, Computer Science Dept., Univ. Toronto, Sept. 1982.

[15] H. Garcia-Molina, "Performance of update algorithms for replicated data in a distributed database," Ph. D. dissertation, Computer Science Dept., Stanford Univ., June 1979.

[16] H. Garcia-Molina and R.K. Abbott, "Reliable distributed database management," *Proc. IEEE*, vol. 75, pp. 601–619, May 1987.

[17] J.N. Gray, "An approach to decentralized computer systems," *IEEE Trans. Software Eng.*, vol. SE-12, pp. 684–692, June 1986.

[18] IBM Corp., "Customer Information Control System / Virtual Storage (CICS/VS): Intercommunication Facilities Guide," SC33–0133, 1983.

[19] J.A. Larson and S. Rahimi, "Tutorial: Distributed database management," IEEE Computer Society Press, 1985.

[20] W. Lin and J. Nolte, "Basic timestamp, multiple version timestamp, and two-phase locking," in *Proc. 9th VLDB Conf.*, Nov. 1983.

[21] R.D. Nelson and B.R. Iyer, "Analysis of a replicated data base," *Performance Evaluation*, vol. 5, pp. 133–148, 1985.

[22] U. Sumita and O.R. Liu Sheng, "Analysis of query processing in distributed database systems with fully replicated files: A hierarchical approach," *Performance Evaluation*, vol. 8, pp. 223–238, 1988.

[23] C. Thanos *et al.*, "The effects of two-phase locking on the performance of a distributed database management system," *Performance Evaluation*, vol. 8, pp. 129–157, 1988.

[24] R. Williams *et al.*, "R*: An overview of the architecture," in *Proc. Int. Conf. Databases*, June 1982, Academic Press, New York, pp. 1–27, 1982.

[25] P.S. Yu *et al.*, "On coupling multisystems through data sharing," *Proc. IEEE*, vol. 75, pp. 573–587, May 1987.

[26] P. S. Yu and D.M. Dias, "Concurrency control using locking with deferred blocking," in *Proc. 6th Int. Conf. Data Engineering*, pp. 30–36, 1990.

[27] P.S. Yu *et al.*, "Modeling and analysis of a time-stamp history based certification protocol for concurrency control," *IEEE Trans. Knowledge and Data Eng.*, vol. 3, pp. 525–537, Dec. 1991.

[28] P. S. Yu *et al.*, "On the analytical modeling of database concurrency control," *J. ACM*, to be published (also available as IBM Research Report RC 15386, Yorktown Heights, NY, Jan. 1991).

[29] P.S. Yu and D.M. Dias, "Analysis of hybrid concurrency control schemes for a high data contention environment," *IEEE Trans. Software Eng.* vol. 18, pp. 118–129, Feb. 1992.

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

From 1987 to 1988, he was a visiting scientist at the IBM Thomas J. Watson Research Center, Yorktown Heights, NY. He is currently with the Computer Science Department of the University of Rome, La Sapienza, Rome, Italy. 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 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.

Philip S. Yu (S'76–M'78–SM'87) received the B.S. degree from National Taiwan University, Taipei, in 1972, and the M.S. and Ph.D. degrees in 1976 and 1978, respectively, all in electrical engineering; he also received the M.B.A. degree from New York University in 1982.

Since 1978 he has been with the IBM Thomas J. Watson Research Center, Yorktown Heights, NY. He is currently 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 systems, parallel and distributed processing, computer architecture, performance modeling, and workload analysis. He has published more than 120 papers in refereed journals and conferences.

Dr. Yu is a member of the ACM, and has served as the program chairman of the Second International Workshop on Research Issues on Data Engineering: Transaction and Query Processing.