Revisiting Commit Processing in Distributed Database Systems

Ramesh Gupta * Jayant Haritsa * Krithi Ramamritham †

* Supercomputer Education and Research Centre
Indian Institute of Science, Bangalore 560012, India
{ramesh, haritsa}@serc.iisc.ernet.in

† Department of Computer Science
Univ. of Massachusetts, Amherst 01003, U.S.A.
krithi@cs.umass.edu

Abstract

A significant body of literature is available on distributed transaction commit protocols. Surprisingly, however, the relative merits of these protocols have not been studied with respect to their *quantitative* impact on transaction processing performance. In this paper, using a detailed simulation model of a distributed database system, we profile the transaction throughput performance of a representative set of commit protocols. A new commit protocol, OPT, that allows transactions to "optimistically" borrow uncommitted data in a controlled manner is also proposed and evaluated. The new protocol is easy to implement and incorporate in current systems, and can coexist with most other optimizations proposed earlier. For example, OPT can be combined with current industry standard protocols such as Presumed Commit and Presumed Abort.

The experimental results show that distributed *commit* processing can have considerably more influence than distributed *data* processing on the throughput performance and that the choice of commit protocol clearly affects the magnitude of this influence. Among the protocols evaluated, the new optimistic commit protocol provides the best transaction throughput performance for a variety of workloads and system configurations. In fact, OPT's peak throughput is often close to the upper bound on achievable performance. Even more interestingly, a *three-phase* (i.e., non-blocking) version of OPT provides better peak throughput performance than all of the standard *two-phase* (i.e., blocking) protocols evaluated in our study.

1 Introduction

Distributed database systems implement a transaction *commit protocol* to ensure transaction atomicity. Over the last two decades, a variety of commit protocols have been proposed by database researchers [4, 20]. These include the classical *two phase commit (2PC)* protocol [13, 17], its variations such as *presumed commit* and *presumed abort* [16, 19], and *three phase commit (3PC)* [25]. To achieve their functionality, these commit protocols typically require exchange of multiple messages, in multiple phases, between the participating sites where the distributed transaction executed. In addition, several log records are generated, some of which have to be "forced", that is, flushed to disk immediately in a synchronous manner. Due to these costs, commit processing can result in a significant increase in transaction execution times [16, 21, 24], making the choice of commit protocol an important design decision for distributed database systems.

In light of the above discussion, it seems reasonable to expect that the results of detailed studies of commit protocol performance would be available to assist distributed database system designers in making an informed choice (as is the case with, for example, distributed *concurrency control* [7, 8, 9]). Surprisingly, however, most of the earlier performance studies of commit protocols (e.g., [1, 16, 19, 21]) have been limited to comparing protocols based on the *number* of messages and the *number* of forced-writes that they incur. Thorough quantitative performance evaluation with regard to *overall* transaction processing metrics such as mean response time or peak throughput has, however, received very little attention. This is a significant lacuna since transaction processing performance is usually a primary concern for database system designers. Hence we investigate the performance of commit protocols in this paper. (The results of a performance study of commit protocols are reported in [14], but in the context of a *client-server* type of DBMS environment. The scope and methodology of their study is also considerably different from ours as explained in [11]. In addition, we present and evaluate a new high-performance protocol (OPT) that is easy to implement and incorporate in current systems.)

Performance Issues

From a performance perspective, commit protocols can be compared on the following three issues:

Effect on Normal Processing: This refers to the extent to which the protocol affects the normal (no-failure) distributed transaction processing performance. That is, how expensive is it to provide atomicity using this protocol?

Resilience to Failures: A commit protocol is said to be *non-blocking* if, in the event of a site failure, it permits transactions that had cohorts executing at the failed site to terminate at the operational sites without waiting for the failed site to recover [20, 25]. With blocking protocols, there is a possibility of transaction processing grinding to a halt in the presence of failures (as explained in Section 2.4). Non-blocking protocols, on the other hand, are designed to ensure that such major disruptions do not occur. To achieve their functionality, however, they usually incur additional messages and forced-writes than their "blocking" counterparts. In general, "two-phase" commit protocols are susceptible to blocking whereas "three-phase" commit protocols are non-blocking [5].

Speed of Recovery: This refers to the time required for the database to be recovered when the failed site comes back up after a crash. That is, how long does it take before transaction processing can commence again in a recovering site?

Of the three issues highlighted above, we believe, from a performance perspective, that the first two issues (effect on normal processing and resilience to failures) are of primary importance since they directly affect ongoing transaction processing. In comparison, the last issue (speed of recovery) appears less critical for two reasons: First, failure durations are usually orders of magnitude larger than recovery times. Second, failures are usually rare enough that we do not expect to see a difference in *average* performance among the protocols because of one commit protocol having a faster recovery time than the other. With this viewpoint, we focus here on the *mechanisms* required during normal operation to provide for recoverability, rather than on the recovery *process* itself.

Contributions

In this paper, we quantitatively investigate the performance implications of supporting distributed transaction atomicity. Our contributions are two-fold:

1. Using a simulator based on a detailed closed queueing model of a distributed database system, we compare the throughput performance of a representative set of previously proposed commit protocols for a variety of distributed database workloads and system configurations. Both blocking (two-phase) commit protocols, and non-blocking (three-phase) commit protocols are included in the scope of our study.

   To isolate and quantify the effects of supporting data distribution and achieving transaction atomicity on system performance, we use two baselines in the simulations: (1) a centralized system, and (2) a system wherein data processing is distributed but commit processing is centralized.

2. We propose and evaluate a new commit protocol, called **OPT**, that, in contrast to earlier commit protocols, allows transactions to "optimistically" borrow dirty (uncommitted) data. Although dirty reads are permitted, there is no danger of incurring cascading aborts [5] since the borrowing is done in a controlled manner. The protocol is easy to implement and to incorporate in current systems, and can be integrated with most other optimizations proposed earlier.

Salient observations from our simulation experiments include:

- Distributed *commit* processing can have considerably more effect than distributed *data* processing on the system performance.

- Among the commit protocols evaluated, the new OPT protocol provided the best overall performance in our experiments, doing considerably better than the classical protocols.[1] In fact, OPT's peak throughput performance was often close to that obtained with the "distributed processing, centralized commit" baseline mentioned above, which in a sense represents an upper bound on achievable performance.

- Due perhaps to their increased overheads, non-blocking protocols such as three-phase commit (3PC) have not been used in real-world systems. However, our experiments show that, under conditions where there is sufficient contention in the system, a combination of OPT and 3PC provides better throughput performance than any of the 2PC-based standard blocking protocols. This suggests that it would be possible for distributed database systems that are operating in high contention situations and are currently using 2PC-based protocols to switch over to OPT-3PC, thereby obtaining the superior performance of OPT during normal processing and, in addition, acquiring the highly desirable non-blocking feature of 3PC. This, in essence, is a "win-win" situation.

- OPT's design is based on the assumption that transactions that lend their uncommitted data will almost always commit. We have found, however, that the performance of OPT is *robust* in that, even if transactions abort in the commit phase (due to violation of integrity constraints, software errors, etc.), OPT maintains its superior performance as long as the probability of such aborts does not exceed *fifteen percent*, a level that is much higher than what might be expected in practice.

2 Distributed Commit Protocols

A common model[2] of a distributed transaction is that there is one process, called the *master*, which is executed at the site where the transaction is submitted, and a set of other processes, called *cohorts*, which execute on behalf of the transaction at the various sites that are accessed by the transaction.[3] For this model, a variety of transaction commit protocols have been devised, most of which are based on the classical **two phase commit (2PC)** protocol [13]. In this section, we briefly describe the 2PC protocol and a few popular variations of this protocol – complete descriptions are available in [19, 25].

2.1 Two Phase Commit Protocol

In this protocol, the master, after receiving a WORKDONE message from all of its cohorts, initiates the first phase of the commit protocol by sending PREPARE (to commit) messages in parallel to all the cohorts. Each cohort that is ready to commit first force-writes a *prepare* log record to its local

[1] A suitably modified version of OPT exhibited similar good performance characteristics in our recent research on commit processing in distributed *real-time* database systems [12].

[2] An alternative "peer-to-peer" model is discussed in [18, 21].

[3] In the most general case, each of the cohorts may itself spawn off sub-transactions at other sites, leading to the "tree of processes" transaction structure of System R* [15] – for simplicity, we only consider a two-level tree here.

stable storage and then sends a YES vote to the master. At this stage, the cohort has entered a *prepared* state wherein it cannot unilaterally commit or abort the transaction but has to wait for the final decision from the master. On the other hand, each cohort that decides to abort force-writes an *abort* log record and sends a NO vote to the master. Since a NO vote acts like a veto, the cohort is permitted to unilaterally abort the transaction without waiting for the final decision from the master.

After the master receives the votes from all the cohorts, it initiates the second phase of the protocol. If all the votes are YES, it moves to a *committing* state by force-writing a *commit* log record and sending COMMIT messages to all the cohorts. Each cohort after receiving a COMMIT message moves to the *committing* state, force-writes a *commit* log record, and sends an ACK message to the master.

If the master receives even one NO vote, it moves to the *aborting* state by force-writing an *abort* log record and sends ABORT messages to those cohorts that are in the prepared state. These cohorts, after receiving the ABORT message, move to the *aborting* state, force-write an *abort* log record and send an ACK message to the master.

Finally, the master, after receiving acknowledgements from all the prepared cohorts, writes an *end* log record and then "forgets" the transaction.

2.2 Presumed Abort

A variant of the 2PC protocol, called **presumed abort (PA)** [19], tries to reduce the message and logging overheads by requiring all participants to follow – at failure recovery time – an "in case of doubt, abort" rule. That is, if after coming up from a failure a site queries the master about the final outcome of a transaction and finds no information available with the master, the transaction is (correctly) assumed to have been aborted. With this assumption, it is not necessary for cohorts to (a) send acknowledgments for ABORT messages from the master, and (b) force-write the *abort* record to the log. It is also not necessary for the master to force-write the *abort* log record or to write an *end* log record after abort.

In short, the PA protocol behaves identically to 2PC for committing transactions, but has reduced message and logging overheads for aborted transactions.

2.3 Presumed Commit

A variation of the presumed abort protocol is based on the observation that, in general, the number of committed transactions is much more than the number of aborted transactions. In this variation, called **presumed commit (PC)** [19], the overheads are reduced for *committing* transactions, rather than aborted transactions, by requiring all participants to follow – at failure recovery time – an "in case of doubt, commit" rule. In this scheme, cohorts do not send acknowledgments for the *commit* global decision, and do not force-write the *commit* log record. In addition, the master does not write an *end* log record. However, the master is required to force-write a *collecting* log record before initiating the two-phase protocol. This log record contains the names of all the cohorts involved in executing that transaction.

The above optimizations of 2PC have been implemented in a number of database products and PA is, in fact, now part of the ISO-OSI and X/OPEN distributed transaction processing standards [18, 21].

2.4 Three Phase Commit

A fundamental problem with all of the above protocols is that cohorts may become *blocked* in the event of a site failure and remain blocked until the failed site recovers. For example, if the master fails after initiating the protocol but before conveying the decision to its cohorts, these cohorts will become blocked and remain so until the master recovers and informs them of the final decision. During the blocked period, the cohorts may continue to hold system resources such as locks on data items, making these unavailable to other transactions, which in turn become blocked waiting for the resources to be relinquished, that is, "cascading blocking" results. It is easy to see that, if the duration of the blocked period is significant, it may result in major disruption of transaction processing activity.

To address the blocking problem, a **three phase commit (3PC)** protocol was proposed in [25]. This protocol achieves a non-blocking capability by inserting an extra phase, called the "precommit phase", in between the two phases of the 2PC protocol. In the precommit phase, a preliminary decision is reached regarding the fate of the transaction. The information made available to the participating sites as a result of this preliminary decision allows a global decision to be made despite a subsequent failure of the master. Note, however, that the price of gaining non-blocking functionality is an increase in the communication overheads since there is an extra round of message exchange between the master and the cohorts. In addition, both the master and the cohorts have to force-write additional log records in the precommit phase.

2.5 Other Protocols

The above-mentioned protocols are well-established and have received the most attention in the literature – we therefore concentrate on them in our study. It should be noted, however, that a variety of other protocols have also been proposed. These include *linear 2PC* [13], *distributed 2PC* [20], *Unsolicited Vote (UV)* [26], *Early Prepare (EP)* and *Coordinator Log (CL)* [22, 23] protocols. Very recently, the *Implicit Yes Vote (IYV)* protocol [1] and the *two-phase abort (2PA)* protocol [3] have been proposed for distributed database systems that are expected to be connected by extremely high speed networks.

3 Optimistic Commit Processing

In all of the protocols described in the previous section, a cohort that reaches the *PREPARED* state can release all of its read locks. However, it has to retain all its *update* locks until it receives the global decision from the master – this retention is fundamentally necessary to maintain atomicity. More importantly, the lock retention interval is *not bounded* since the time duration that a cohort is in the *PREPARED* state can be arbitrarily long (for example, due to network delays). If the retention period is large, it may have a significant negative effect on performance since other transactions that wish to access this (prepared) data are forced to block until the commit processing is over. It is important to note that this *data blocking* is *orthogonal* to the *decision blocking* (because of failures) that was discussed in Section 2.4. That is, in all the commit protocols, including 3PC, transactions can be affected by prepared data blocking. Moreover, such data blocking occurs during normal processing whereas decision blocking only occurs during failure situations.

To address the above issue of (prepared) data blocking, we have designed a new version of the 2PC protocol, in which transactions requesting data items held by other transactions in the prepared state are allowed to access this data. That is, prepared cohorts *lend* uncommitted data to concurrently executing transactions. Given such lending, two situations may arise:

Lender Receives Decision First :
Here, the lending cohort receives its global decision before the borrowing cohort has completed its local execution. If the global decision is to commit, the lending cohort completes its processing in the normal fashion. If the global decision is to abort, the lender is aborted in the normal fashion; in addition, the borrower is also aborted since it has utilized inconsistent data.

Borrower Completes Execution First :
Here, the borrowing cohort completes its execution before the lending cohort has received its global decision. The borrower is now "put on the shelf", that is, it is made to wait and not allowed to send a WORKDONE message to its master. This means that the borrower is not allowed to initiate the processing that could eventually lead to its reaching the *prepared* state. Instead, it has to wait until the lender receives its global decision. If the lender commits, the borrower is "taken off the shelf" and allowed to send its WORKDONE message. However, if the lender aborts, the borrower is also aborted immediately since it has utilized inconsistent data.

In summary, the protocol allows transactions to access uncommitted data held by prepared transactions in the "optimistic" belief that this data will eventually be committed. We will hereafter refer to this protocol as **OPT**.

3.1 Aborts in OPT do not Cascade

An important point to note here is that OPT's policy of using uncommitted data is generally *not* recommended in database systems since this can potentially lead to the well-known problem of *cascading aborts* [5] if the transaction whose dirty data has been accessed is later aborted. However, for the OPT protocol, this problem is alleviated due to two reasons:

1. The lending transaction is typically expected to commit because (a) the lending cohort is in prepared state and cannot be aborted due to local data conflicts, and (b) the sibling cohorts are also expected to eventually vote to commit since they have survived all their data conflicts that occurred prior to the initiation of the commit protocol. In fact, if we assume that a locking-based concurrency control mechanism such as 2PL [10] is used, it is easy to verify that there is *no* possibility of sibling cohorts aborting, during the commit processing period, due to serializability considerations. Therefore, an abort vote can arise only due to other reasons such as, violation of integrity constraints, software errors, system failure, etc. We will hereafter use the term "surprise" aborts to refer to this type of aborts.

2. Even if the lending transaction does eventually abort, it only results in the abort of the immediate borrower and does not cascade beyond this point (since the borrower is not in the prepared state – the only situation in which uncommitted data can be accessed). In short, the abort chain is bounded and is of length one (of course, if an aborting lender has lent to multiple borrowers, then all of them will be aborted, but the length of each abort *chain* is limited to one).

3.2 Integrating Prior 2PC Optimizations with OPT

Apart from the optimistic data access described above, the following features can also be included in the OPT protocol:

Presumed Abort/Commit : The optimizations of Presumed Commit or Presumed Abort discussed earlier for 2PC can also be used in conjunction with OPT to reduce the protocol overheads. We consider both options in our experiments.

Nonblocking OPT : Our description of OPT above assumed a 2PC protocol as the basis. However, the OPT approach can be applied directly to the 3PC protocol as well. We evaluate the performance of this protocol also in our experiments.

Other Optimizations : Apart from PA and PC, a variety of other optimizations have been proposed for the 2PC protocol. A comprehensive description and analysis of such optimizations is presented in [21]. Among these, the optimizations we have examined are *Read-Only* (one phase commit for read-only transactions), *Unsolicited Vote* (cohorts enter prepared state and vote yes without waiting for a prepare request from coordinator), *Long Locks* (cohorts piggyback their commit acknowledgments onto subsequent messages), *Shared Logs* (cohorts share a common log with the master), *Group Commit* (forced writes are batched together to save on disk I/O), and *linear 2PC* [13] (message overheads are reduced by ordering the sites in a linear chain for communication purposes).

OPT is especially attractive to integrate with protocols such as *3PC*, *Group Commit* and *linear 2PC*, since they *extend* the period during which data is held in the prepared state. However, when combined with protocols such as *Unsolicited Vote* and *IYV* – which do not guarantee that a cohort which has unilaterally entered the prepared state will not be forced back later into an active state – OPT can lead to cascading aborts, long "on-the-shelf"-times for borrowers, deadlocks involving the lender and the borrower, etc. [11]. Barring these exceptions, virtually all of the above optimizations can be integrated with an OPT implementation to produce *enhanced* performance.

3.3 System Integration

We now comment on the implementation issues related to the OPT protocol:

1. The lock manager at each site must be modified to permit borrowing of data held by prepared cohorts.

2. The lock manager must keep track of the cohorts that have borrowed prepared data so that, if the lender aborts, the borrowers can also be aborted.

3. For a borrower cohort that finishes execution before its lenders have received their global decision, the local transaction manager must not send a WORKDONE message until the fate of its lenders is determined.

The above modifications do not appear difficult to incorporate in current database system software. Moreover, as shown later in our experiments, the performance benefits that can be derived from these changes suggest that it is worthwhile to make the effort of implementing them.

4 Simulation Model

To evaluate the performance of the various commit protocols described in the previous sections, we developed a detailed simulator based on a closed queueing model of a distributed database system. Our simulation model is similar to the one used in [7] to study distributed concurrency control protocols. A summary of the key model parameters is given in Table 1.

The database is modeled as a collection of $DBSize$ pages that are uniformly distributed across all the $NumSites$ sites. At each site, the transaction multiprogramming level is specified by the MPL parameter. Each transaction in the workload has the "single master – multiple cohort" structure described in Section 2. The number of sites at which each transaction executes is specified by the $DistDegree$ parameter. The master and one cohort reside at the site where the transaction is submitted whereas the remaining $DistDegree - 1$ cohorts are set up at different sites chosen at random from the remaining $NumSites - 1$ sites. At each of the execution sites, the number of pages accessed by the transaction's cohort varies uniformly between 0.5 and 1.5 times $CohortSize$. These pages are chosen randomly from among the database pages located at that site. A page that is read is updated with probability $UpdateProb$. A transaction that is aborted is restarted after a delay and makes the same data accesses as its original incarnation. The length of the delay is equal to the average transaction response time – this is the same heuristic as that used in most transaction management studies [2, 7, 8, 9]. After a transaction completes, a new one is submitted immediately at its originating site.

The physical resources at each site consist of $NumCPUs$ processors, $NumDataDisks$ data disks, and $NumLogDisks$ log disks. The data disks store the data pages while the log disks store the transaction log records. There is a single common queue for the processors whereas each of the disks has its own queue. All queues are processed in an FCFS order except that message processing is given higher priority than data processing at the CPUs. The $PageCPU$ and $PageDisk$ parameters capture the CPU and disk processing times per data page, respectively. For simplicity, we assume that all data is accessed from disk and buffer pool considerations are therefore ignored.

The communication network is simply modeled as a switch that routes messages since we assume a local area network that has high bandwidth. However, the CPU overheads of message transfer, given by the $MsgCPU$ parameter, are taken into account at both the sending and the receiving sites. This means that there are two classes of CPU requests – local data processing requests and message processing requests, and as noted above, message processing is given higher priority than data processing.

4.1 Transaction Execution

When a transaction is initiated, it is assigned the set of sites where it has to execute and the data pages that it has to access at each of these sites. The master is then started up at the originating site, forks off a local cohort and sends messages to initiate each of its cohorts at the remote participating sites. Transactions in a distributed system can execute in either sequential or parallel fashion. The distinction is that cohorts in a sequential transaction execute one after another, whereas cohorts in a parallel transaction are started together and execute independently until commit time. We consider both types of transactions in our study.

Each cohort makes a series of read and update accesses. A read access involves a concurrency control request to obtain access permission, followed by a disk I/O to read the page, followed by a period of CPU usage for processing the page. Update requests are handled similarly except for their disk I/O – the writing of the data pages takes place asynchronously after the transaction has committed. We assume sufficient buffer space to allow the retention of data updates until commit time. The commit protocol is initiated when the transaction has completed its data processing.

Table 1: Simulation Model Parameters

$NumSites$	Number of sites in the database
$DBSize$	Number of pages in the database
MPL	Transaction multiprogramming level / site
$TransType$	Transaction Type (Sequential or Parallel)
$DistDegree$	Degree of Distribution (number of cohorts)
$CohortSize$	Average cohort size (in pages)
$UpdateProb$	Page update probability
$NumCPUs$	Number of processors per site
$NumDataDisks$	Number of data disks per site
$NumLogDisks$	Number of log disks per site
$PageCPU$	CPU page processing time
$PageDisk$	Disk page access time
$MsgCPU$	Message send / receive time

4.2 Concurrency Control

For transaction concurrency control (CC), we use the distributed strict two-phase locking (2PL) protocol [5]. Transactions, through their cohorts, set read locks on pages that they read and update locks on pages that need to be updated. All locks are held until the receipt of the PREPARE message from the master. Subsequently, the cohort releases all its read locks but retains its update locks until it receives and implements the global decision from the master. For the new protocol, OPT, however, the lock manager at each site is modified to permit borrowing of updated data items held by prepared transactions.

With respect to deadlocks, in our simulation implementation, both global and local deadlock detection is immediate, that is, a deadlock is detected as soon as a lock conflict occurs and a cycle is formed. The youngest transaction in the cycle is restarted to resolve the deadlock. We do not explicitly model the overheads for detecting deadlocks or for concurrency control since (a) these costs would be similar across all the commit protocols, and (b) they are usually negligible compared to the overall cost of accessing data [7].

Another point to note here is that, as mentioned in Section 3.1, with this CC mechanism, there is no possibility of serializability-induced aborts occurring in the *commit processing* stage.

4.3 Logging

With regard to logging costs, we explicitly model only *forced* log writes since they are done synchronously and suspend transaction operation until their completion. The cost of each forced log write is the same as the cost of writing a data page to the disk.

5 Experiments and Results

Using the distributed database model described in the previous section, we conducted an extensive set of simulation experiments comparing the performance of the various commit protocols presented in Sections 2 and 3. Due to space limitations, we discuss only a representative set of results here – the complete details are available in [11].

The primary performance metric of our experiments is *transaction throughput*, that is, the rate at which the system completes transactions.[4] We also emphasize the *peak* throughput that is achievable by each protocol since this represents the maximum attainable performance and by using a suitable admission control policy (for example, Half-and-Half [6]), the throughput can be maintained at this level in high-performance systems. All the throughput graphs of this paper show mean values that have relative half-widths about the mean of less than 10 percent at the 90 percent confidence level, with each experiment having been run until at least 50000 transactions were processed by the system. Only statistically significant differences are discussed here.

5.1 Comparative Protocols

To help isolate and understand the effects of distribution and atomicity on throughput performance, and to serve as a basis for comparison, we have also simulated the performance behavior of two additional scenarios:

Centralized System :
In this scenario, hereafter referred to as **CENT**, a *centralized* database system that is equivalent (in terms of database size and physical resources) to the distributed database system is modeled. Messages are obviously not required here and commit processing only requires force-writing a *single* decision log record. Modeling this scenario helps to isolate the overall effect of distribution on throughput.

Distributed Processing, Centralized Commit :
In this scenario, hereafter referred to as **DPCC**, data processing is executed in the normal distributed fashion, that is, involving messages. The *commit* processing, however, is like that of a centralized system, requiring only the force-writing of the decision log record at the master. While this system is clearly artificial, modeling it helps to isolate the effect of distributed commit processing on throughput (as opposed to the centralized scenario which eliminates the entire effect of distributed processing).

5.2 Experiment 1: Resource and Data Contention

The settings of the workload parameters and system parameters for our first experiment are listed in Table 2. These

Table 2: Baseline Parameter Settings

$NumSites$	8	$NumCPUs$	1
$DBSize$	8000 pages	$NumDataDisks$	2
$TransType$	Parallel	$NumLogDisks$	1
$DistDegree$	3	$PageCPU$	5 ms
$CohortSize$	6 pages	$PageDisk$	20 ms
$UpdateProb$	1.0	$MsgCPU$	5 ms

settings were chosen to ensure significant levels of both resource contention (RC) and data contention (DC) in the system, thus helping to bring out the performance differences between the various commit protocols.

In this experiment, each transaction executes in a *parallel* fashion at three sites, accessing and updating an average of six pages at each site. Each site has a single CPU, two data disks and one log disk. The CPU and disk processing times are such that the system operates in an I/O-bound region. However, since it is not heavily I/O-bound, it is possible for message-related CPU costs to shift the system into a region of CPU-bound operation – this occurs, for example, in Experiment 4 where a higher degree of transaction distribution, and consequently more network activity, is modeled.

For this experiment, Figure 1a presents the transaction throughput results as a function of the per-site multiprogramming level. We first observe here that for all the protocols, the throughput initially increases as the MPL is increased, and then it decreases. The initial increase is due to the fact that better performance is obtained when a site's CPUs and disks are utilized in parallel. As the multiprogramming level is increased beyond a certain level, however, the system starts thrashing due to data contention, resulting in degraded transaction throughput.

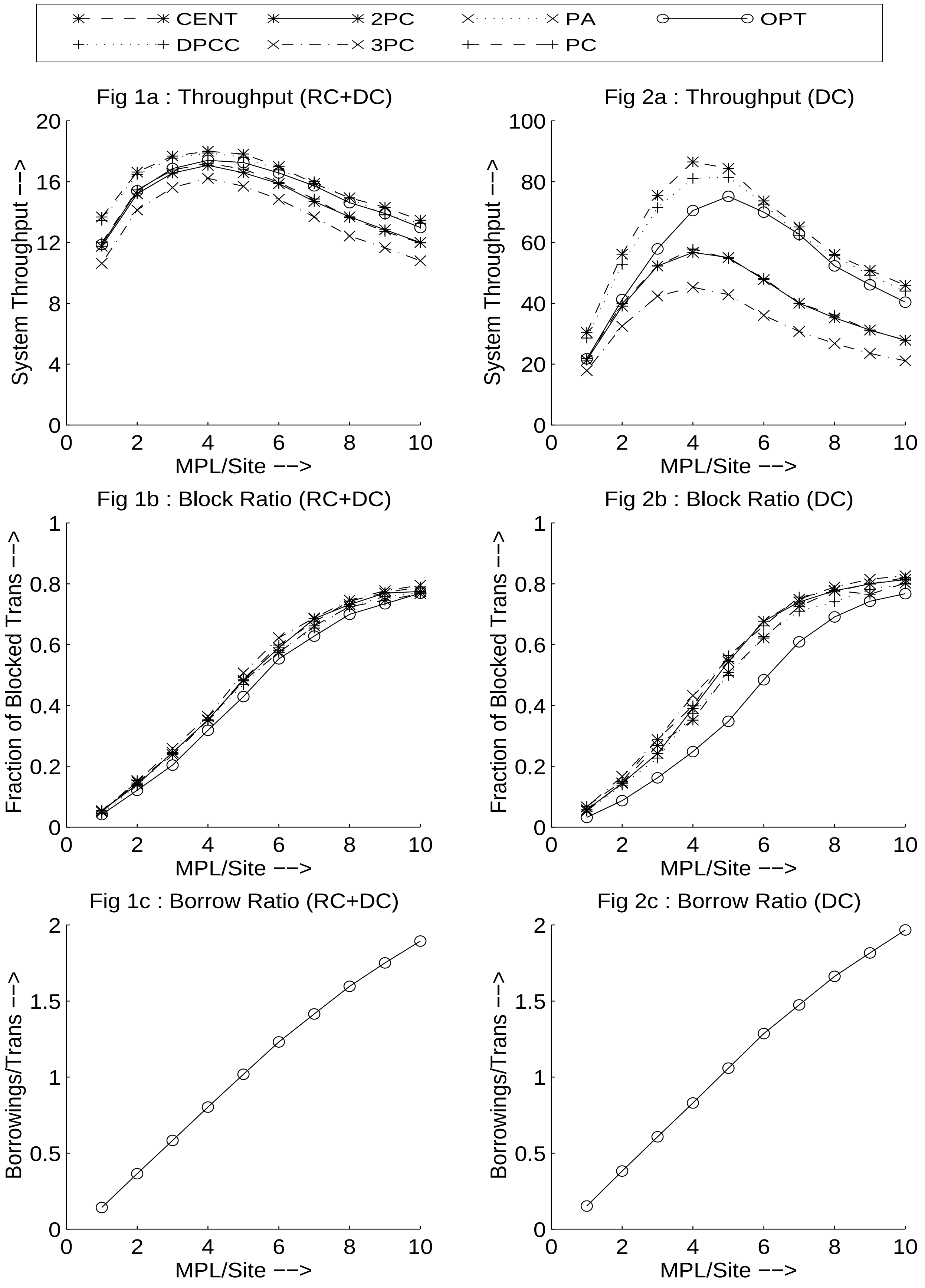
Comparing the relative performance of the protocols in Figure 1a, we observe that the centralized system (CENT) performs the best, as expected, and that the performance of distributed processing/centralized commit (DPCC) is very close to that of CENT. In contrast, there is a noticeable difference between the performance of these baseline systems and the performance of the classical commit protocols (2PC, PA, PC, 3PC) throughout the loading range. This demonstrates that distributed *commit* processing can have considerably more effect (difference between the performance of DPCC and 2PC) than distributed *data* processing (difference between the performance of CENT and DPCC) on the throughput performance, as will be seen in our other experiments as well.

Moving on to the relative performance of 2PC and 3PC, we observe that there is a clear difference in their performance. The difference arises from the additional message and logging overheads involved in 3PC, which are shown in Table 3 (for committing transactions).

Table 3: Protocol Overheads (DistDegree = 3)

Protocol	Execution	Commit	
	Messages	Forced-Writes	Messages
2PC	4	7	8
PA	4	7	8
PC	4	5	6
3PC	4	11	12
DPCC	4	1	0
CENT	0	1	0

[4]Since we are using a closed queueing model, the inverse relationship between throughput and response time (as per Little's Law) makes either a sufficient performance metric.


CENT
2PC
PA
OPT
DPCC
3PC
PC
Fig 1a : Throughput (RC+DC)
System Throughput -->
MPL/Site -->
Fig 2a : Throughput (DC)
System Throughput -->
MPL/Site -->
Fig 1b : Block Ratio (RC+DC)
Fraction of Blocked Trans -->
MPL/Site -->
Fig 2b : Block Ratio (DC)
Fraction of Blocked Trans -->
MPL/Site -->
Fig 1c : Borrow Ratio (RC+DC)
Borrowings/Trans -->
MPL/Site -->
Fig 2c : Borrow Ratio (DC)
Borrowings/Trans -->
MPL/Site -->

Shifting our focus to PC, we observe that it performs very similarly to 2PC. This is because of the following reasons: First, while PC saves on the forced writes and acknowledgements at each cohort, it incurs an additional *collecting* forced write at the master. Second, the forced writes that PC saves were done in parallel at the cohorts by 2PC. Therefore, even though PC saves considerably on the overheads it fails to appreciably reduce the *overall* response time of the transaction. Thus the gain in throughput for PC over 2PC is not significant.

There is no possibility of serializability-induced aborts in the commit phase, as mentioned earlier in Section 4.2, due to using distributed strict 2PL as the CC mechanism. In the absence of any other source of aborts, as in this experiment, PA reduces to 2PC and performs *identically*. We will therefore defer further discussion of the performance of PA until Experiment 6 where we model surprise transaction aborts in the commit phase.

Finally, turning our attention to the new protocol, OPT, we observe that its performance is either the same or better than that of all the standard algorithms over the full range of MPL. At low MPLs, when there is less data contention, and consequently little opportunity for borrowing, OPT is virtually identical to 2PC and therefore performs at the same level. At higher MPLs, however, the performance of OPT is superior to that of 2PC, and in fact, becomes close to that of DPCC. The reason for this is that in OPT, the cohorts in the *PREPARED* state do not contribute to the data contention thus reducing the number of blocked transactions. This is seen in Figure 1b, which shows the transaction "block ratio", that is, the average fraction of transactions that are in the blocked state. Finally, in Figure 1c, we graph the "borrow ratio" for OPT, that is, the average number of data items (pages) borrowed per transaction. This graph clearly shows that borrowing comes into the picture at higher MPLs, resulting in improved performance for OPT.

5.3 Experiment 2: Pure Data Contention

The goal of our next experiment was to isolate the influence of *data contention* (DC) on the performance of the commit protocols. For this experiment, the physical resources (CPUs and disks) were made "infinite", that is, there is no queueing for these resources [2]. The other parameter values are the same as those used in Experiment 1. The results of this experiment are shown in Figures 2a through 2c. In these figures, as in the previous experiment, CENT shows the best performance and the performance of DPCC is close to that of CENT. The performance of the standard protocols relative to the baselines is, however, markedly worse than before. This is because in Experiment 1, the considerable difference in overheads between CENT and 2PC was largely submerged due to the resource and data contention in the system having a predominant effect on transaction response times. In the current experiment, however, where throughput is limited only by data contention, although transaction response times are typically smaller than under RC+DC, the commit phase here occupies a bigger *proportion* of the overall transaction response time and therefore the overheads of 2PC are felt to a greater extent. Similarly, 3PC performs significantly worse than 2PC due to its considerable extra overheads. Finally, the performance of PC remains similar to that of 2PC, for the same reasons as those given in the previous experiment.

Moving on to OPT, we observe that at low MPLs, it behaves almost identically to 2PC since there are few opportunities for borrowing. At higher MPLs, however, OPT's performance is substantially better than that of 2PC. In fact, OPT's peak throughput is close to that of DPCC. An interesting observation here is that 2PC, DPCC and CENT all achieve their peak throughput at an MPL of 4, while for OPT, the corresponding MPL value is 5. This is explained by considering Figure 2b, which shows the transaction block ratio, where we see that this ratio is significantly lower for OPT than the other protocols for the same multiprogramming level due to the elimination of blocking for prepared data. Thus for a given data contention level, OPT allows more concurrency in the system than standard protocols. Finally, Figure 2c presents the transaction borrow ratio, again showing that borrowing increases almost linearly with MPL.

In summary, this experiment indicates that under high data contention, OPT's performance can be substantially superior to that of the standard protocols.

5.4 Experiment 3: Fast Network Interface

In the previous experiments, the cost for sending and receiving messages modeled a system with a relatively slow network interface ($MsgCpu = 5\ ms$). We conducted another experiment wherein the network interface was faster by a factor of five, that is, $MsgCpu = 1\ ms$. The experiment was conducted for both resource-cum-data contention (RC+DC) and pure data contention (DC) scenarios.[5] For brevity, we only discuss the results of this experiment here – the graphs are available in [11]: First, the performance of all the protocols becomes closer to that of CENT as compared to the previous experiments, and in fact, DPCC and CENT are virtually indistinguishable. This improved behavior of the protocols is only to be expected since low message costs effectively eliminate the effect of a significant fraction of the overheads involved in each protocol. Under pure DC, however, the remaining overheads of forced-writes are significant enough to point out clear differences between the performances of DPCC and 2PC and those of 2PC and 3PC. Second, OPT's peak throughput performance is again close to that of DPCC in both the RC+DC and pure DC cases.

This experiment shows that adopting the OPT principle can be of value even with very high-speed network interfaces because faster message processing does not necessarily eliminate the *data contention* bottleneck.

5.5 Experiment 4: Higher Degree of Distribution

In the experiments described so far, each transaction executed on *three* sites. To investigate the impact of having a higher degree of distribution, we performed an experiment wherein each transaction executed on *six* sites. The *CohortSize* in this experiment was reduced from 6 pages to 3 pages in order to keep the average transaction length equal to that of the previous experiments. For this environment, the overheads of the protocols are shown in Table 4 (for committing transactions).

For this experiment, Figures 3a and 3b present the transaction throughput results under RC+DC and under pure DC, respectively. In Figure 3a, we observe that the performance of the baselines, CENT and DPCC, is virtually indistinguishable although the message processing overheads

[5]The default parameter settings for the RC+DC and pure DC scenarios in this experiment, as well as in the following experiments, are the same as those used in Expt. 1 and Expt. 2, respectively.

Table 4: Protocol Overheads (DistDegree = 6)

Protocol	Execution	Commit	
	Messages	Forced-Writes	Messages
2PC	10	13	20
PA	10	13	20
PC	10	8	15
3PC	10	20	30
DPCC	10	1	0
CENT	0	1	0

of DPCC are significantly more. The reason for this is that even with these additional overheads, the CPU is still not the bottleneck for DPCC, resulting in its performance being similar to that of CENT (as in Experiment 1).

For the standard protocols, however, the increase in the message overheads is sufficiently large (Table 4) that the system now operates in a heavily *CPU-bound* region. As a result, the performance differences between the baselines and these protocols are visibly more than those seen for the limited distribution workload of Experiment 1 (Figure 1a).

An interesting feature of Figure 3a is that, for the *first* time, we observe a significant difference between the performance of PC and that of 2PC. In fact, PC exhibits better performance than 2PC across the entire MPL range. The reason for this behavior is the heavily CPU-bound nature of the workload. In this environment, PC's reduced overheads result in its performing better than 2PC. This is further confirmed in the results of Figure 3b, where in the absence of resource contention, PC again performs almost identically to 2PC.

Turning our attention to OPT, we observe that under RC+DC (Figure 3a), the performance of OPT is only marginally better than that of 2PC. This is due to two factors: (i) the CPU-bound nature of the workload, and (ii) message processing has higher priority over data processing. As a result of these factors, the increase in execution phase length is much larger than the increase in commit phase length, resulting in a smaller "commit-execution ratio". Since OPT's impact is felt only during the commit phase, the decrease in this ratio causes reduced performance improvement as compared to Experiment 1. These results may seem to suggest that OPT may not be the best approach for workloads of the type considered in this experiment. Note, however, that we can now bring into play the ability of OPT to "peacefully and usefully coexist" with other optimizations by combining OPT and PC to form an **OPT-PC** protocol. The performance of this protocol is also shown in Figure 3a, wherein it provides the best overall performance due to deriving the benefits of both the OPT and PC optimizations.

Under pure data contention (Figure 3b), note that the performance difference between CENT and DPCC is more as compared to earlier experiments. This is because the average transaction response time is reduced due to the heightened degree of distribution. As a result, message delays now form a significant fraction of the response time. We also observe in Figure 3b that the difference between the performance of DPCC and that of 2PC is now very large (the peak throughput of DPCC is more than *twice* that of 2PC). This again clearly demonstrates that distributed commit processing can affect performance to a significantly greater degree than distributed data processing.

As mentioned earlier, PC performs almost the same as 2PC in Figure 3b. The performance of OPT continues to be superior to 2PC, but in contrast to what was seen in Figure 3a, we now observe that the performance of OPT-PC is no better than that of OPT at low MPLs and slightly worse at higher MPLs. This is because the effect of OPT-PC is to increase the length of the execution phase of the transaction (by introducing *collecting* forced write) and at the same time to reduce the commit phase length (by saving on forced writes at cohorts), thus decreasing the commit-execution ratio and diminishing the utility of the optimistic feature.

5.6 Experiment 5: Non-Blocking OPT

In the previous experiments, we observed that OPT, which is based on 2PC, performed significantly better than the standard protocols. This motivated us to evaluate the effect of incorporating the same optimizations in the **3PC** protocol. We refer to this protocol as OPT-3PC and for this experiment, Figures 4a and 4b present the throughput results under RC+DC and pure DC, respectively. In Figure 4a we observe that the performance of OPT-3PC is similar to that of 3PC at lower MPLs. However, at higher MPLs, OPT-3PC not only performs better than 3PC, but also achieves a peak throughput that is comparable to that of 2PC. These observations show up more vividly in Figure 4b, where the peak throughput of OPT-3PC actually significantly *surpasses* that of 2PC.

An important point to note here is that the optimistic feature has potentially more impact in the 3PC context than in the 2PC context. This is because the length of the prepared state is significantly longer in 3PC (due to the extra phase), thereby increasing the benefits of borrowing.

In summary, these results indicate that by using the OPT approach, we can *simultaneously* obtain both the highly desirable non-blocking functionality and better peak throughput performance than the classical blocking protocols. This, in essence, is a "win-win" situation.

5.7 Experiment 6: Surprise Aborts

In all of the experiments discussed earlier, there was no potential for serializability-induced aborts in the commit processing stage, as explained in Section 4.2. It is for this reason that no difference was possible in the relative performance of PA and 2PC. In practice, however, aborts may arise in the commit phase due to other reasons such as violation of integrity constraints, software errors, system failure, etc. We therefore conducted an experiment where such "surprise abort" situations were modeled and evaluated their impact on protocol performance. The important point to note here is that these situations are fundamentally biased against OPT since its underlying assumption that transactions will usually commit may not hold under surprise aborts.

In this experiment, each cohort, on receiving the PREPARE message from the master, instead of always voting YES, randomly votes NO with a certain probability. We consider three different cases, where the probability of a cohort aborting arbitrarily is 1 percent, 5 percent and 10 percent, respectively. Since each transaction is composed of three cohorts, these cohort abort probabilities translate to overall *transaction* abort probabilities of approximately 3 percent, 15 percent and 27 percent, respectively. Note that the latter two cases model abort probabilities that appear artificially high as compared to what might be expected in practice. Modeling these high abort levels, however, helped us determine the extent to which OPT was *robust*.

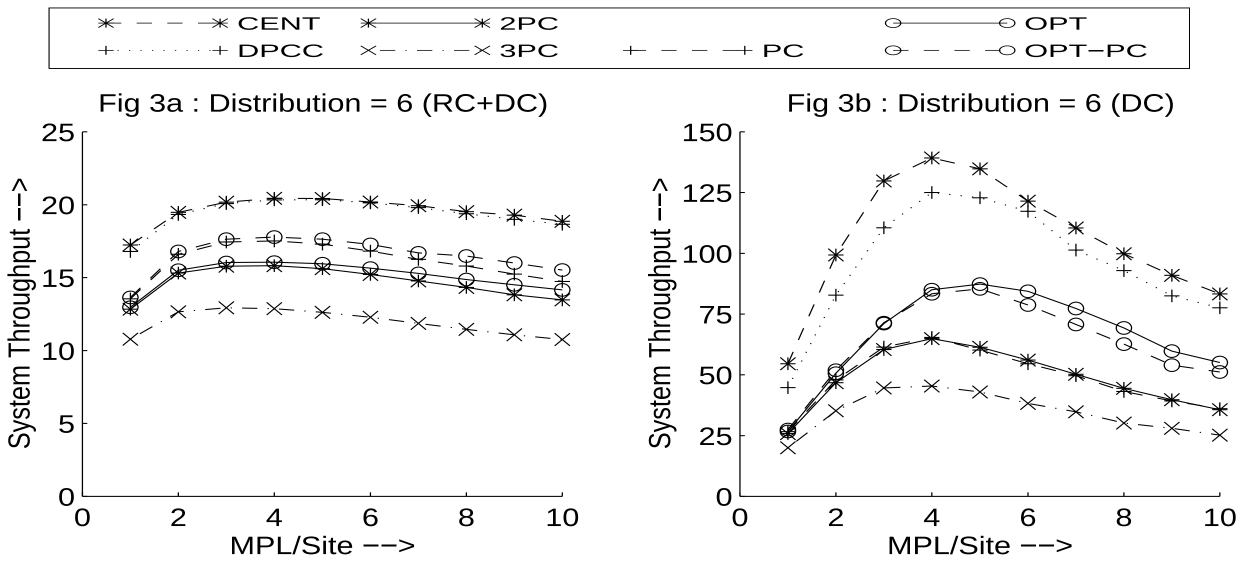


Fig 3a : Distribution = 6 (RC+DC)

Fig 3b : Distribution = 6 (DC)

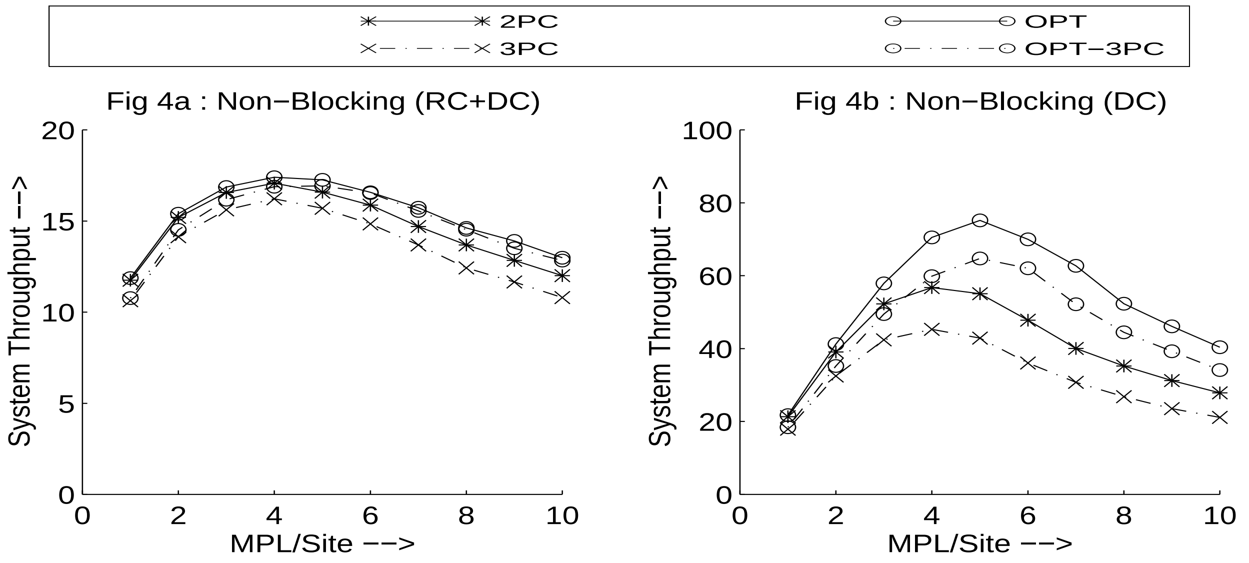


Fig 4a : Non−Blocking (RC+DC)

Fig 4b : Non−Blocking (DC)

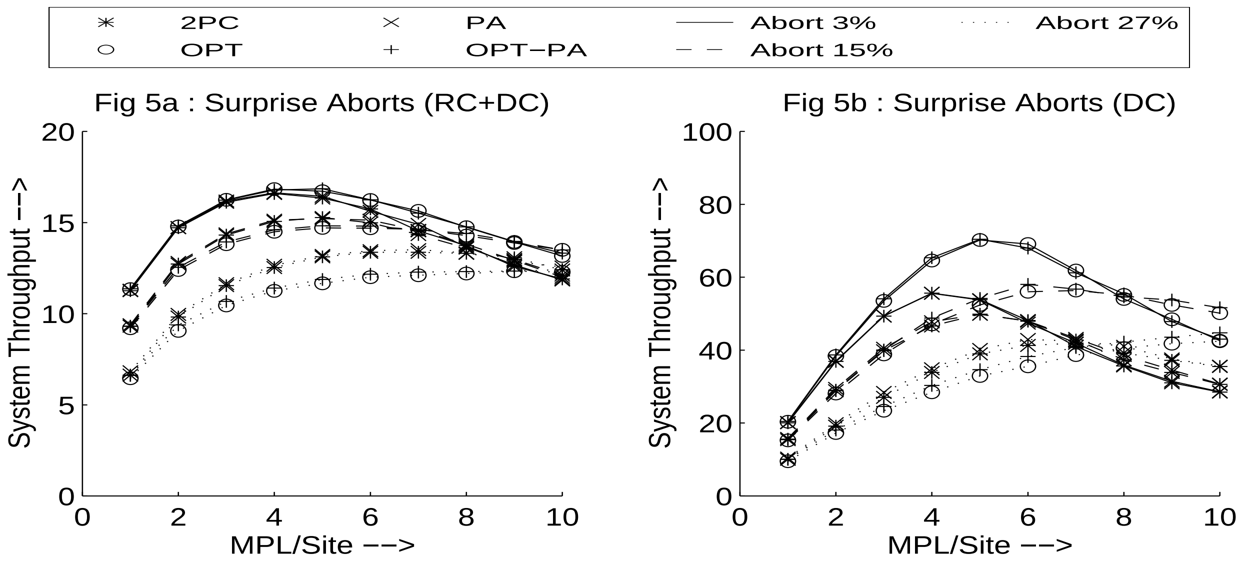


Fig 5a : Surprise Aborts (RC+DC)

Fig 5b : Surprise Aborts (DC)

The results of this experiment under RC+DC and under pure DC are presented in Figures 5a and 5b, respectively. We observe here that OPT's peak throughput performance is comparable to that of 2PC even when 15 percent of the transactions are aborting and it is only when the abort level is in excess of this value that we begin to see an appreciable difference in performance. At these high abort levels, the optimism of borrowing transactions proves to be misplaced since they are often aborted due to the abort of their lenders.

We also observe in these figures that, in spite of having a large fraction of aborts, PA which is designed to do well in the abort case, shows only marginal improvement over 2PC. This is because although PA saves on the forced writes and acknowledgements, these savings are small in comparison to the *total* overheads of commit processing. For example, in the 27 percent transaction abort probability case, 2PC incurs about 8.8 forced writes and 2.5 acknowledgements per committed transaction, whereas the corresponding values for PA are 7.7 and 2, respectively. When the system is not heavily resource-bound, these savings do not affect the throughput performance significantly. To investigate this further, we conducted the same experiment with a higher degree of distribution ($DistDegree = 6$), which results in a heavily CPU-bound environment, as in Experiment 4. In this situation, we found that the savings by PA were sufficient to make it perform clearly better than 2PC.

The important point to note here is that OPT can *also* derive these performance benefits of PA by combining with it to form OPT-PA – the performance of this combined algorithm is also presented in Figures 5a and 5b and shows the expected outcome.

Another interesting feature in Figures 5a and 5b is that, at high MPLs, the performance of the OPT and 2PC protocols in a system with higher probability of surprise aborts is better than their performance in a system with lower probability of aborts, that is, there is a *performance crossover*. The reason for the crossover is the following: In our model, as in earlier transaction management studies (for example, [2]), aborted transactions are delayed before restarting with the delay period being equal to the average response time. This delay effectively becomes a crude way of controlling the data contention in the system. At high MPLs, this results in the system with higher transaction abort probabilities having less data contention than the system with lower transaction abort probabilities and therefore performing better, resulting in the crossover. Note, however, that the *peak* throughput is significantly smaller for the system with higher transaction abort probabilities than the one with lower abort probabilities.

5.8 Other Experiments

We conducted several other experiments to explore various regions of the workload space. These included workloads with sequential transactions, reduced update probabilities, small database sizes, etc. The relative performance of the protocols in these additional experiments remained qualitatively similar to that seen in the experiments described here (see [11] for details), with the performance improvement delivered by OPT being dependent on the level of data contention in the system.

One interesting feature that we noticed in the sequential transaction experiments is that the performance differences between the protocols *decreased* as compared to carrying out the same experiment with parallel transactions. This is because the length of the execution phase is more for sequential transactions as compared to their parallel counterparts. The length of the *commit* phase, however, is the same for both types of transactions. Hence, the commit-execution ratio is significantly smaller in the case of sequential transactions (this effect was confirmed by measuring these values in our experiments), resulting in OPT having lesser impact on the throughput.

6 Conclusions

In this paper, we have quantitatively investigated the performance implications of supporting distributed transaction atomicity. Using a detailed simulation model of a distributed database system, we evaluated the throughput performance of a variety of standard commit protocols, including 2PC, PA, PC and 3PC, apart from two baseline protocols, CENT and DPCC. We also developed and evaluated a new commit protocol, OPT, that was designed specifically to reduce the blocking arising out of locks held on prepared data. To the best of our knowledge, these are the first quantitative results on commit processing in a fully-distributed DBMS with respect to end-user performance metrics.

Our experiments demonstrated the following:

1. Distributed *commit* processing can have considerably more effect than distributed *data* processing on the system performance. This highlights the need for developing high-performance commit protocols.

2. The PA and PC variants of 2PC, which reduce protocol overheads, have been incorporated in a number of database products and transaction processing standards. In our experiments, however, we found that these protocols provide tangible benefits over 2PC only in a few restricted situations. PC performed well only when the degree of distribution was high – in current applications, however, the degree of distribution is usually quite low [21]. PA, on the other hand, performed only marginally better than 2PC even when the probability of surprise aborts was close to thirty percent – in practice, surprise aborts occur only occasionally.

   We caution the reader here that the above conclusion is limited to the completely update transaction workloads considered here – PA and PC have additional optimizations for fully or partially *read-only* transactions [19] that may have a significant impact in workloads having a large fraction of these kinds of transactions.

3. Use of the new protocol, OPT, either by itself, or in conjunction with other standard optimizations, leads to significantly improved performance over the standard algorithms for all the workloads and system configurations considered in this paper. Its good performance is attained primarily due to its reduction of the blocking arising out of locks held on prepared data. A feature of the OPT protocol design is that it limits the abort chain to one, thereby preventing cascading aborts. This is achieved by allowing only prepared transactions to lend their data and ensuring that borrowers cannot enter the prepared state until their borrowing is terminated.

   The power of the optimistic access feature is substantial enough that OPT's throughput performance was often close to that obtained with the DPCC baseline, which represents an upper bound on achievable performance.

4. An attractive feature of OPT is that it can be integrated, often synergistically, with most other optimizations proposed earlier. In particular, in the few experiments wherein PC or PA performed noticeably better than 2PC, using OPT-PC or OPT-PA resulted in the best performance. The only exception wherein OPT does not appear to combine well is with protocols that allow cohorts to enter the prepared state unilaterally and therefore run the risk of having to revert to an active state in case the master subsequently sends additional work to the cohort (for example, the Unsolicited Vote protocol of distributed Ingres [26]).

5. OPT's design is based on the assumption that transactions that lend their uncommitted data will almost always commit. However, OPT is reasonably robust in that it maintains its superior performance until the probability of surprise transaction aborts in the commit phase exceeds as high a value as fifteen percent. Beyond this level, its performance becomes progressively worse than that of the classical protocols.

6. Under conditions where there is sufficient contention in the system, a combination of OPT and 3PC provides better throughput performance than the standard 2PC-based blocking protocols. This suggests that it would be possible for distributed database systems that are operating in high contention situations and currently using 2PC-based protocols to switch over to OPT-3PC, thereby obtaining the superior performance of OPT during normal processing and, in addition, acquiring the highly desirable non-blocking feature of 3PC.

In summary, we suggest that distributed database systems currently using the 2PC, PA or PC commit protocols may find it beneficial to switch over to using the corresponding OPT algorithm, that is, OPT, OPT-PA or OPT-PC. If having non-blocking functionality is important but 3PC has not been used due to its excessive overheads resulting in poor performance, then OPT-3PC appears to be an attractive choice since it provides a peak throughput that exceeds those of the standard 2PC protocols.

Acknowledgements

This work was supported in part by research grants from the Dept. of Science and Technology, Govt. of India and from the National Science Foundation of the United States under grant IRI - 9314376.

References

[1] Y. Al-Houmaily and P. Chrysanthis, "Two-Phase Commit in Gigabit-Networked Distributed Databases", *Proc. of 8th Intl. Conf. on Parallel and Distributed Computing Systems*, September 1995.

[2] R. Agrawal, M. Carey and M. Livny, "Concurrency Control Performance Modeling: Alternatives and Implications", *ACM Trans. on Database Systems*, 12(4), 1987.

[3] S. Banerjee and P. Chrysanthis, "A Fast and Robust Failure Recovery Scheme for Shared-Nothing Gigabit-Networked Databases", *Proc. of 9th Intl. Conf. on Parallel and Distributed Computing Systems*, September 1996.

[4] B. Bhargava, (editor), *Concurrency and Reliability in Distributed Database Systems*, Van Nostrand Reinhold, 1987.

[5] P. Bernstein, V. Hadzilacos and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, 1987.

[6] M. Carey, S. Krishnamurthi and M. Livny, "Load Control for Locking: The 'Half-and-Half' Approach", *Proc. of 9th Symp. on Principles of Database Systems*, April 1990.

[7] M. Carey and M. Livny, "Distributed Concurrency Control Performance: A Study of Algorithms, Distribution, and Replication", *Proc. of 14th Intl. Conf. on Very Large Data Bases*, August 1988.

[8] M. Carey and M. Livny, "Parallelism and Concurrency Control Performance in Distributed Database Machines", *Proc. of ACM SIGMOD Conf.*, June 1989.

[9] M. Carey and M. Livny, "Conflict Detection Tradeoffs for Replicated Data", *ACM Trans. on Database Systems*, 16(4), 1991.

[10] K. Eswaran et al, "The Notions of Consistency and Predicate Locks in a Database Systems", *Comm. of ACM*, 19(11), 1976.

[11] R. Gupta, J. Haritsa and K. Ramamritham, "Revisiting Commit Processing in Distributed Database Systems", *TR-97-01*, DSL/SERC, Indian Institute of Science, 1997.

[12] R. Gupta, J. Haritsa, K. Ramamritham and S. Seshadri, "Commit Processing in Distributed Real-Time Database Systems", *Proc. of 17th IEEE Real-Time Systems Symposium*, December 1996.

[13] J. Gray, "Notes on Database Operating Systems", *Operating Systems: An Advanced Course, Lecture Notes in Computer Science*, 60, 1978

[14] M. Liu, D. Agrawal and A. El Abbadi, "The Performance of Two-Phase Commit Protocols in the Presence of Site Failures", *Proc. of 24th Intl. Symp. on Fault-Tolerant Computing*, June 1994.

[15] B. Lindsay et al, "Computation and Communication in R*: A Distributed Database Manager", *ACM Trans. on Computer Systems*, 2(1), 1984.

[16] B. Lampson and D. Lomet, "A New Presumed Commit Optimization for Two Phase Commit", *Proc. of 19th Intl. Conf. on Very Large Data Bases*, August 1993.

[17] B. Lampson and H. Sturgis, "Crash Recovery in a Distributed Data Storage System", *Tech. Report*, Xerox Palo Alto Research Center, 1976.

[18] C. Mohan and D. Dievendorff, "Recent Work on Distributed Commit Protocols, and Recoverable Messaging and Queuing" *Data Engineering Bulletin*, 17(1), 1994.

[19] C. Mohan, B. Lindsay and R. Obermarck, "Transaction Management in the R^* Distributed Database Management System", *ACM Trans. on Database Systems*, 11(4), 1986.

[20] M. Ozsu and P. Valduriez, *Principles of Distributed Database Systems*, Prentice-Hall, 1991.

[21] G. Samaras, K. Britton, A. Citron and C. Mohan, "Two-Phase Commit Optimizations in a Commercial Distributed Environment", *Journal of Distributed and Parallel Databases*, 3(4), 1995 (also in *Proc. of 9th IEEE Intl. Conf. on Data Engineering*, April 1993).

[22] J. Stamos and F. Cristian, "A Low-Cost Atomic Commit Protocol", *Proc. of 9th Symp. on Reliable Distributed Systems*, October 1990.

[23] J. Stamos and F. Cristian, "Coordinator Log Transaction Execution Protocol", *Journal of Distributed and Parallel Databases*, 1(4), 1993.

[24] P. Spiro, A. Joshi and T. Rengarajan, "Designing an Optimized Transaction Commit Protocol", *Digital Technical Journal*, 3(1), 1991.

[25] D. Skeen, "Nonblocking Commit Protocols", *Proc. of ACM SIGMOD Conf.*, June 1981.

[26] M. Stonebraker, "Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES", *IEEE Trans. on Software Engg.*, 5(3), 1979.