Adding PEP to Real-Time Distributed Commit Processing

Jayant R. Haritsa*
Database Systems Lab, SERC
Indian Institute of Science
Bangalore 560012, INDIA
haritsa@dsl.serc.iisc.ernet.in

Krithi Ramamritham†
Dept. of CSE
Indian Institute of Technology
Mumbai 400076, INDIA
krithi@cse.iitb.ernet.in

Abstract

One-phase commit protocols substantially reduce the overheads of transaction commit processing, making them attractive for incorporation in distributed real-time databases. A major obstacle, however, is that these protocols significantly increase the occurrence of priority inversions. This arises from the cohorts of each distributed transaction being in a prepared state for extended periods of time, during which their data locks cannot be pre-empted.

We present here a new one-phase real-time commit protocol, called PEP, that addresses the above-mentioned problem by suitably utilizing our previously proposed prepared data borrowing technique. Simulation-based evaluation of PEP for real-time applications with firm deadlines demonstrates that, for a variety of environments, it substantially reduces the number of killed transactions as compared to its multi-phase counterparts. In fact, PEP often provides better performance than even an equivalent centralized system.

1. Introduction

Developing high-performance transaction commit protocols is an important problem in the design of distributed real-time database systems (**DRTDBS**). This is because commit processing forms a significant component of transaction execution times [10], arising from the several messages that have to be exchanged between the participating sites where the distributed transaction executes, as also the writing of several log records, some of which have to be "forced", that is, flushed to disk immediately.

We took a first step towards addressing the above issue in a recent paper [5], wherein we investigated the real-time performance of a variety of commit protocols, including the classical Two-Phase Commit (2PC) [3] and Three-Phase Commit (3PC) [12]. We also proposed a new 2PC-based commit protocol called PROMPT and showed that, for firm-deadline [4] applications, it was far more successful than the classical protocols in minimizing the number of "killed" (permanently aborted due to missing their deadlines) transactions.

Our earlier studies focussed exclusively on two and three-phase commit protocols. In these protocols, commit processing begins only after *all* the data processing associated with a transaction has been completed. We move on, in this paper, to investigating the real-time characteristics of *one-phase commit* (1PC) protocols. There is no explicit "voting phase" to decide on the outcome (commit or abort) of the transaction in these protocols since cohorts enter the "prepared state" at the time of sending the work completion message itself – that is, the commit processing and data processing activities are effectively *overlapped*. By eliminating an entire phase and thereby reducing commit processing overheads and durations, 1PC protocols seem potentially capable of completing more transactions before their deadlines. In this paper, we evaluate the real-time performance of Early Prepare (EP) [13], a representative 1PC protocol.

A fundamental problem with real-time commit processing is that cohorts that are in the prepared state cannot be pre-empted, even if a higher priority transaction requires their data, resulting in priority inversion. In 1PC protocols, the insidious effects of priority inversion are exacerbated because of the overlap between data processing and commit processing, which means that a cohort can enter the prepared state long before the transaction has completed its full data processing. There is hence the possibility that the advantages gained from eliminating a protocol phase may be negated by the ill-effects of increased priority inversion.

To address the above problem, we incorporate the technique of controlled borrowing of prepared data [5], developed originally for PROMPT, in the Early Prepare framework. The resulting protocol, called **PEP** (for PROMPT-

*Part of this work was done during a sabbatical at Lucent Bell Labs, 600 Mountain Avenue, Murray Hill, NJ 07974, USA.
†Also affiliated with Dept. of Computer Science, Univ. of Massachusetts, Amherst, MA 01003, USA.

EP), thus has features designed to reduce the negative effects of both (a) multiple rounds of communication and logging overheads, and (b) priority inversion due to the data of prepared cohorts being inaccessible until the associated transaction commits or aborts.

Using a detailed simulator of a DRTDBS, we evaluate the performance of the EP and PEP 1PC protocols over a variety of real-time transaction workloads and system configurations. We compare their performance with that of PROMPT, the state-of-the-art in real-time commit processing. Our evaluation suite also includes a baseline system called CENT, which models a fully centralized database, and therefore has no communication or logging overheads at all – modeling this scenario helps to isolate the performance impact of distribution. To the best of our knowledge, the results presented here represent the first work on one-phase real-time commit processing.

2. Background: The PROMPT Protocol

In this section, we provide background material about the PROMPT (Permits Reading Of Modified Prepared-data for Timeliness) real-time commit protocol, which we developed in our previous research [5]. For ease of exposition, the following notation is used in the sequel – "SMALL CAPS FONT" for messages, "`typewriter font`" for log records, and "sans serif font" for transaction states.

We assume, in the following discussion, the subtransaction model of a distributed transaction wherein a *master* process executes at the user-site and a set of *cohort* processes execute on behalf of the master at the remote execution sites. The master assigns work to each cohort via a STARTWORK message and a cohort sends a WORKDONE message to the master after it has completed its assigned work.

The classical 2PC protocol [3] fails, with respect to meeting (firm) real-time objectives, on two related counts: First, by making prepared data inaccessible, it increases transaction blocking times and therefore has an adverse impact on the number of killed transactions. Second, since it does not take transaction priorities into account, priority inversion may result and cause the affected high-priority transactions to miss their deadlines.

The above problems are alleviated in PROMPT, which is structurally similar to 2PC, by allowing controlled access to prepared data. That is, prepared cohorts *lend* their uncommitted data to concurrently executing transactions so that these transactions can continue their local data processing. Later, if the global decision for the transaction associated with the lending cohort is to commit, the lending cohort commits in the normal fashion; further, if the borrowing cohort has already completed its local data processing, a WORKDONE message is sent to its master. On the other hand, 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 dirty data.

In short, the PROMPT protocol allows transactions to access uncommitted data held by prepared transactions in the "optimistic" belief that this data will eventually be committed. To further improve its real-time performance, a variety of additional optimizations are also included in the PROMPT protocol. The most important of these is *Active Abort*: In the basic 2PC protocol, cohorts are "passive" in that they inform the master of their status only upon explicit request by the master. This is acceptable in conventional distributed DBMS since, after a cohort has completed its data phase, there is *no possibility* of the cohort subsequently being aborted due to serializability considerations (assuming a locking-based concurrency control mechanism). In a DRTDBS, however, a cohort which is not yet in its commit phase can be aborted due to conflicts with higher priority transactions. Therefore, it may be better for an aborting cohort to immediately inform the master so that the abort of the transaction at the sibling sites can be done earlier. Early restarts are beneficial in two ways: First, they provide more time for the restarted transaction to complete before its deadline. Second, they minimize the wastage of both logical and physical system resources. Accordingly, cohorts in PROMPT follow an "active abort" policy – they inform the master as soon as they decide to abort locally.

3. The PEP Real-Time Commit Protocol

We now move on to presenting our new PEP (Prompt–Early Prepare) one-phase real-time commit protocol. PEP represents an integration of the 1PC approach of the EP protocol and the prepared data lending philosophy of PROMPT. The main performance-related features of PEP are the following: First, from its 1PC basis, it incurs decreased logging and message overheads. Second, from PROMPT it inherits the ability for the controlled lending of prepared data. Third, it has two additional features that occur without any explicit steps being taken on the part of PEP's commit process, that is, these are inherent in PEP's design. One is the occurrence of "active aborts", described earlier in Section 2, and the other is the interesting and beneficial feature whereby the longer a transaction executes, the lower is its probability of being aborted due to data conflicts. These features are explained in detail below.

3.1. One-phase Commit Process

PEP adopts the one-phase commit approach of the Early Prepare protocol [13], wherein the first phase of traditional two-phase commit processing is overlapped with the data processing phase. To achieve this, cohorts in PEP force a

Table 1. Message Overheads Comparison

Outcome	Master→Cohort		Cohort→Master		Reduction
	PEP	PROMPT	PEP	PROMPT	per cohort
Commit	1	2	0	2	3
Abort	1	2	1	2	2

Table 2. Logging Overheads Comparison

Outcome	MASTER			
	Total Log Records		ForcedWrites	
	PEP	PROMPT	PEP	PROMPT
Commit	2	2	2	1
Abort	2	2	1	1

Outcome	COHORT			
	Total Log Records		ForcedWrites	
	PEP	PROMPT	PEP	PROMPT
Commit	2	2	1	2
Abort	2	2	2	2

`prepare` log record to the local disk and enter the prepared state at the time of sending the WORKDONE message itself. This means that the voting phase becomes redundant since the master implicitly knows, when it receives a WORKDONE message, that the associated cohort is prepared to commit. Hence, only the decision phase is required in this protocol.

An additional optimization to further reduce the overheads is to use the Presumed Commit (PC) [9] mechanism wherein all participants follow – at failure recovery time – an "in case of doubt, commit" rule. With this scheme, cohorts do not send ACKs for a commit decision sent from the master, and also do not force-write the `commit` log record. In addition, the master does not write an `end` log record. To make recovery possible with this mechanism, the master force-writes a `membership` log record at transaction initiation time, which lists the identifiers of all the cohorts involved in executing that transaction.

With the above features, PEP's commit processing overheads are substantially reduced with respect to PROMPT, as summarized in Tables 1 and 2 for both commit and abort outcomes. Note that PEP reduces the communication overhead with respect to PROMPT by at least 2 messages *per cohort*, irrespective of the outcome. With regard to logging, although the master in PEP has an additional forced-write compared to PROMPT, each cohort has to do one less, so overall PEP will have fewer forced-writes. Further, even the additional forced-write at the master can be eliminated by using more sophisticated forms of PC, such as New PC [8] – for simplicity, we only consider classical PC here.

3.2. Lending of Prepared Data

The above-mentioned decrease in resource overheads of PEP does not come for free, however, in the real-time environment: It can significantly increase the impact of priority inversion due to its extension of the duration of the prepared state of the individual cohorts. While in PROMPT (and all other two-phase protocols), the prepared state for a cohort begins only after the global transaction has completed its data processing, in PEP (as in EP) it begins as soon as a cohort has finished its *local* data processing. Further, whether a distributed transaction is *sequential* or *parallel* now becomes an issue – in a sequential transaction cohorts execute one after another, whereas in a parallel transaction the cohorts execute concurrently. With PEP, the average duration of the prepared state for cohorts of sequential transactions is *significantly higher* as compared to that in (equivalent) parallel transactions. This is because, in the sequential case, each cohort remains in the prepared state during the entire data processing of all the cohorts that are executed after it, whereas in the parallel case, each cohort remains in this state only during the interval between its own completion and that of its slowest sibling cohort.

The heightened inversion possibilities may result in increased blocking times for high-priority transactions, resulting in missing their deadlines and adversely impacting the system performance. To address this issue, we can utilize the same lending technique described for PROMPT in the previous section, whereby prepared cohorts lend their uncommitted data to concurrently executing transactions so that these transactions can continue their local data processing. In fact, the lending possibilities are much greater in PEP than in PROMPT precisely because many more conflicts have to be resolved by lending, rather than by the conflict resolution mechanism (e.g. *priority abort* [1]) in operation.

However, these increased lending possibilities lead to the following two problems:

Harmful Lendings: Unlike PROMPT where the chances of a lender subsequently aborting are small (since the lending is for a relatively short duration and the transaction has already fully completed its data processing), these chances are potentially greater in PEP because

1. lending durations are a function of the overall length of the transaction in the case of sequential distributed transactions, and a function of the length of the slowest cohort in the the case of parallel distributed transactions, and
2. the remaining unfinished data processing of the parent transaction may result in aborts arising out of data conflict resolution.

This means that there may be more lendings that are harmful to system performance since they result in the aborts of all the associated borrowers and therefore should be avoided.

Deadlocks: While deadlocks between borrowers and lenders could never occur in the PROMPT framework (again due to the lending becoming operational only after a transaction has fully completed its data processing), deadlocks are an issue with PEP since transactions may continue to access new data items even after having lent some of their previously accessed data.

We have developed specialized schemes for addressing the above problems – due to space limitations, however, we do not discuss them here, but refer the reader to [6]. The performance results presented in this paper are based on a "bare-bones" (i.e., conservative) implementation of PEP that does not incorporate these special schemes. So, nothing is done to reduce harmful lendings while deadlocks are resolved by the simple expedient of allowing them to continue until one of the transactions in the cycle is killed due to deadline expiry. We expect PEP's performance to improve further with our new schemes – in our current work, we are exploring these choices.

3.3. Inherent Active Aborts by Cohorts

In PROMPT, the cohorts have to be explicitly designed to follow the "active abort" policy, i.e., inform the master as soon as they decide to abort locally. In contrast, a very attractive feature of PEP is that this policy is *inherent* in its design, arising out of its EP foundation: When a cohort is aborted due to a conflict with higher priority transactions, it immediately informs its master that it is aborting. So, while a cohort can be locally aborted during its data processing, it cannot abort *after* sending its WORKDONE message. This means that a cohort can respond to a master's STARTWORK message with either a WORKDONE message or an ABORT message, but it is never the case that an ABORT message can follow the sending of the WORKDONE message.

3.4. Inherent Progress-based Priority Increase

Normally, under a priority-based conflict resolution policy, when a higher priority transaction needs a data item locked in a conflicting mode by a lower priority transaction, it can cause the abort of the lower priority transaction. An interesting effect of PEP's basis in EP is that the longer a transaction executes the *lesser* are its chances of being aborted, independent of its actual priority. This is because a cohort cannot be aborted as a result of the data items accessed by any of its prior cohorts (in the sequential case) and by any of its previously completed cohorts (in the parallel case), since all these cohorts are already in the prepared state. In short, for transactions in PEP, the "past is safe, only the present and future are uncertain".

The positive fallout of the above feature is that PEP derives – for free – the following well-established recommendation in the RTDBS literature [1, 7]: Transactions that have already completed a significant portion of their work should be given preferential treatment since their restart at this late stage would waste a lot of already invested resources. For example, in [7], priority restart is used as the conflict resolution mechanism in the early stages of a transaction's execution, wherease priority inheritance is used in the later stages. The transition point is based on the number of locks acquired by the transaction. In contrast to this essentially *boolean* decision, with PEP there is a more *gradual* and continuous increase in preference with progress (arising out of the increasing number of cohorts that become non-preemptable), which is desirable.

Viewed differently, essentially what PEP does is to effectively "tune" the priority basis of the conflict resolution protocol. For example, if 2PL-High Priority [1] is in use (as in our simulation study), PEP starts with 2PL-High Priority, but implicitly moves it closer and closer towards pure (unprioritized) 2PL as the transaction makes more and more progress. Further, this feature comes fully integrated with the commit protocol.

3.5. Comparison with PROMPT

A pictorial representation of the differences between PEP and PROMPT with respect to their commit processing and lending policies is shown in Figures 1a and 1b for sequential and parallel distributed transaction execution, respectively. The intervals during which lending is possible are also marked in the figures (the symbol **P** signifies the cohort's entering the prepared phase). As is evident from these figures, the lending possibilities and lending periods are much more for PEP as compared to PROMPT. Further, the commit processing overhead is significantly smaller for PEP.

The highlights of the PEP protocol were presented in this section. For the complete details of the mechanics of the PEP protocol, we refer the reader to [6].

4. Simulation Model

To evaluate the performance of the commit protocols, we used the simulation model of our previous commit processing study [5], which consists of a (nonreplicated) database that is distributed over a set of sites connected by a network. A summary of the model parameters and their default settings are given in Table 3. In the remainder of this section,

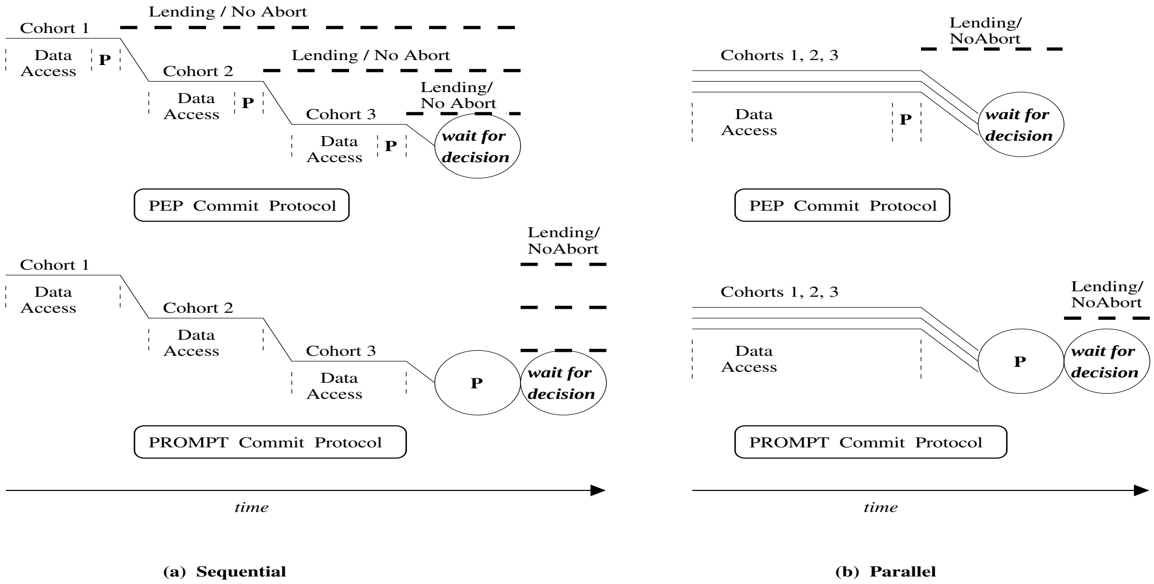

Figure 1. Comparison of the PEP and PROMPT protocols

Table 3. Simulation Model Parameters

Parameter Name	Parameter Meaning	Default Value
$DBSize$	No. of pages in the database	2400
$NumSites$	No. of sites in the database	8
$ArrivalRate$	Transactions/second per site	0 - 10
$SlackFactor$	Slack Factor in Deadline	4.0
$TransType$	Execution Pattern	Parallel
$DistDegree$	Degree of Distribution	3
$CohortSize$	Average cohort size	6 pages
$UpdateProb$	Page update probability	0.5
$NumCPUs$	No. of processors per site	2
$NumDataDisks$	No. of data disks per site	3
$NumLogDisks$	No. of log disks per site	1
$PageCPU$	CPU page processing time	5 ms
$PageDisk$	Disk page access time	20 ms
$MsgCPU$	Message send / receive time	5 ms
$BufHit$	Probability of a buffer hit	0.1

we highlight the main features of this model – the complete details are in [5].

Database and Workload Model. The database is modeled as a collection of $DBSize$ pages that are uniformly distributed across all the $NumSites$ sites. At each site, transactions arrive in an independent Poisson stream with rate $ArrivalRate$, and each transaction has an associated firm deadline. The deadline is assigned using the formula, $D_T = A_T + SF * R_T$, where D_T, A_T and R_T are the deadline, arrival time and resource time, respectively, of transaction T, while SF is a slack factor. The resource time is the total service time at the resources that the transaction requires for its execution. The $SlackFactor$ parameter is a constant that provides control over the tightness/slackness of transaction deadlines.

All transactions have 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 uniformly (without replacement) from among the database pages located at that site. A page that is read is updated with probability $UpdateProb$. A transaction that is aborted due to a data conflict is restarted immediately and makes the same data accesses as its original incarnation.

System Model. The physical resources at each site consist of $NumCPUs$ processors, $NumDataDisks$ data disks, and $NumLogDisks$ log disks. There is a single common queue for the CPUs and the service discipline is Pre-emptive Resume, with preemptions being based on transaction priorities. Each of the disks has its own queue and is scheduled according to a Head-Of-Line (HOL) policy, with the request queue being ordered by transaction priority. The $PageCPU$ and $PageDisk$ parameters capture the CPU and disk processing times per data page, respectively. The $BufHit$ parameter gives the probability of finding a requested page already resident in the buffer pool.

The communication network is simply modeled as a switch that routes messages as a local area network that has high bandwidth is assumed. However, the CPU overheads of message transfer, given by the $MsgCPU$ parameter, are taken into account at both the sending and the receiving sites.

Priority Assignment. Transactions are assigned priorities using the *Earliest Deadline First* (EDF) policy, and cohorts inherit their parent transaction's priority. Further, this priority which is assigned at arrival time, is maintained throughout the course of the transaction's existence in the system.

Concurrency Control. For transaction concurrency control, an extended version of the classical centralized *2PL High Priority (2PL-HP)* protocol [1] is used. The extensions made for operating in our DRTDBS environment are the following: First, when a cohort enters the prepared state, it releases all its read locks but retains the update locks until it receives and implements the global decision from the master.[1] Second, a cohort that is in the prepared state cannot be aborted, irrespective of its priority. Third, in the PROMPT and PEP commit protocols, cohorts in the data-processing phase are allowed optimistic access to data held by conflicting prepared cohorts.

Logging. With regard to logging costs, only *forced* log writes are explicitly modeled since they are done synchronously and suspend transaction operation until their completion.

Transaction Execution Model. At arrival time, a transaction is assigned the set of sites where it has to execute, the data pages that it has to access at each of these sites, and its deadline. The parameter $TransType$ specifies whether the transaction will execute in a sequential or parallel fashion. The master is then started up at the originating site, which in turn forks off a local cohort and sends messages to initiate each of its cohorts at the remote participating sites. 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 also 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. If the transaction's deadline expires either before this point, or before the master has written the global decision log record, the transaction is killed (the precise semantics of firm deadlines in a distributed environment are defined in [5]).

[1]This release does not cause consistency violations with the subtransaction model of transaction execution [6].

Performance Metric. The performance metric of our experiments is **KillPercent**, which is the steady-state percentage of input transactions that are killed, i.e., the percentage of input transactions that the system is *unable* to complete before their deadlines.[2]

Baseline Protocol. To help isolate and understand the effects of data distribution on KillPercent performance, we have also simulated the performance behavior for **CENT** (Centralized), a *centralized* database system that is equivalent (in terms of overall database size and number of physical resources) to the distributed database system. Messages are obviously not required here and commit processing only requires writing a *single* decision log record (force-write if the decision is to `commit`).

5. Experiments and Results

Using the firm-deadline DRTDBS model described in the previous section, we have conducted an extensive set of simulation experiments comparing the real-time performance of the PEP, EP, PROMPT and CENT protocols. In this section, we present the results of a representative set of these experiments. We examine both sequential and parallel execution of subtransactions, a wide range of transaction arrival rates, environments with bounded and unbounded physical resources, distributed systems with different communication delays, and finally, different transaction distribution scenarios. For ease of exposition, we first present results for parallel distributed transactions, and then move on to sequential distributed transactions.

5.1. Parallel Transactions

5.1.1 Experiment 1: Resource and Data Contention

Our first experiment was conducted using the default settings for all model parameters (Table 3), resulting in significant levels of both resource contention (**RC**) and data contention (**DC**). Here, each transaction executes in a parallel fashion at three sites, accessing an average of six pages at each site. An accessed page is updated with a probability of 0.5. Each site has two CPUs, three data disks and one log disk. For this environment, Figure 2a shows the KillPercent behavior as a function of the transaction arrival rate.

Here, we first observe that the EP protocol, although it has no real-time specific features and suffers from significant priority inversion, actually does slightly better than the carefully designed real-time PROMPT throughout the loading range! This highlights the substantial impact that just

[2]The KillPercent values shown here have relative half-widths about the mean of less than 10% at the 90% confidence level – each experiment was run until at least 20000 transactions were processed by the system.

the elimination of one phase and the consequent reduction of overheads can have on real-time commit performance.

Moving on to PEP, we find that by virtue of its lending feature, it largely addresses the priority inversion limitation of EP, while retaining its positive overhead reduction feature, and therefore performs *significantly better* than PROMPT. For example, at an arrival rate of 2.0 transactions per second, PEP has a kill percentage of 2 while PROMPT has close to 5. In fact, PEP's performance is so good that it actually comes *close to CENT*, the centralized system, for most of the loading range. This testifies to PEP's efficient integration of the 1PC and prepared data lending policies.

5.1.2 Experiment 2: Pure Data Contention

In our next experiment, we isolated the influence of *data contention* (DC) on the performance of the commit protocols. Studying this scenario is important because while resource contention can usually be addressed by purchasing more and/or faster resources, there do not exist any equally simple mechanisms to reduce data contention.

For this experiment, the physical resources (CPUs and disks) were made "infinite", that is, there is no queuing for these resources. The other parameter values are the same as those used in Experiment 1. The KillPercent performance results for this system are presented in Figure 2b (note that the Y-axis extends only from 0 to 10 percent). In this graph, we see that, similar to the previous experiment, EP performs similar to or marginally better than PROMPT throughout the loading range. PEP's performance gain is even more striking than before. For example, at an arrival rate of 5 transactions per second, PEP misses about 2.5 percent of transaction deadlines whereas PROMPT misses 10 percent.

The increased improvement in PEP's performance can be attributed to the fact that because of the abundant resources available, transaction restarts do not have a performance impact in terms of resource wastage. Therefore, there are only negligible negative effects due to more lenders aborting in PEP as compared to PROMPT.

5.1.3 Experiment 3: Fast Network Interface

In the previous experiments, the cost model for sending and receiving messages assumed a system with a relatively slow network interface ($MsgCpu = 5ms$). We conducted another experiment wherein the network interface was faster by a factor of five, that is, where $MsgCpu = 1ms$. Due to space limitations, we only discuss the results here without presenting the associated graphs – they are available in [6].

For the RC+DC environment, the qualitative performance of the protocols was similar to that seen in the previous experiments, but all the protocols came quantitatively closer to each other. In the corresponding Pure DC environment, the results were similar except that we found, for the first time, PROMPT outperforming EP.

The above results are only to be expected since the effect of the reduction in overheads that EP brought to bear is reduced due to the improved communication infrastructure, and therefore PROMPT's real-time features prevail. However, PEP whose goal is to combine the benefits of both EP and PROMPT continues to provide the best performance.

5.1.4 Experiment 4: Highly Distributed Transactions

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 experiments 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. In addition, a higher value of $SlackFactor = 6$ was used to cater to the increase in response times caused by the increased distribution. The results for this experiment are shown in Figure 3a for the RC+DC environment, and in Figure 3b for the Pure DC environment.

In Figure 3a, we see a considerable gap between the performance of PEP and EP as compared to that of PROMPT. For example, at an arrival rate of 3 transactions per second, PROMPT misses over 30 percent of transaction deadlines, where PEP brings this down by an order of magnitude, to less than 3 percent (EP has a kill percentage of about 5 percent).

The reason for this dramatic change is that increased distribution results in a substantial relative increase in commit processing overheads for PROMPT, whereas it has comparatively less effect on PEP and EP due to their one-phase commit. This is quantitatively shown in Table 4, which compares the overheads of the protocols for committing transactions with $DistDegree = 3$ and $DistDegree = 6$, respectively (the numbers have been computed as per our simulation model where one cohort is local to the master site, thereby involving no messages). In particular, in the earlier experiments, PROMPT was using 3 extra forced-writes and 6 more messages than PEP per committed transaction, whereas in the current experiment, it uses 6 extra forced-writes and 15 more messages than PEP to achieve the same goal.

Table 4. Protocol Overheads

Protocol	DistDegree = 3		DistDegree = 6	
	ForcedWrites	Messages	ForcedWrites	Messages
PROMPT	7	8	13	20
PEP, EP	5	2	8	5

5.2. Sequential Transactions

Having established the good performance of PEP in the parallel distributed transaction environment, we now turn our attention to sequential distributed transactions. For this, all of the experiments described above for parallel transactions were re-run with sequential cohort execution, and their results are discussed below.

5.2.1 Experiment 5: Resource and Data Contention

In Figure 4a, which presents the results for the baseline RC + DC environment, we see that the results are generally similar to those of the equivalent parallel scenario (Experiment 1), with EP performing similar to PROMPT and PEP being much better than both of them. A major difference, however, is that we now see PEP clearly outperforming CENT! For example, at an arrival rate of 2.5 transactions per second, PEP misses less than 3 percent of the deadlines while CENT misses over 6 percent.

At first sight, the above result may appear to be strange, given that CENT has virtually no commit processing overheads at all. The explanation for the observed counter-intuitive behavior lies in the interaction between PEP and the 2PL-HP concurrency control protocol. As mentioned in Section 3, PEP essentially transforms the classical 2PL-HP protocol into an *adaptive* algorithm that is 2PL-HP at the beginning of the transaction execution but progressively becomes closer to pure 2PL as more and more cohorts of the transaction complete their data processing. This desirable effect comes into play much more in the sequential domain because the data processing of the individual cohorts occur in sequence whereas in the parallel domain, they are overlapped (as shown in Figure 1).

The CENT protocol, on the other hand, has to live with 2PL-HP for the *entire* duration of each transaction, and therefore does not derive the benefit that PEP receives due to its adaptive influence on 2PL-HP. This interesting result highlights the power of the 1PC approach, which goes far beyond merely the obvious reduction of commit processing overheads.

5.2.2 Experiment 6: Pure Data Contention

In Figure 4b, which refers to the baseline Pure DC environment, we see that the performance is similar to that of Experiment 2, but with the following important difference: We now have PROMPT performing noticeably better than EP throughout the loading range. The reason for this behavior is that in the sequential environment, the negative effects of EP's extended priority inversion periods have more impact since the durations of these inversion periods are significantly longer as compared to the parallel environment. PEP, however, continues to provide the best performance by virtue of its lending policy negating these drawbacks of the 1PC approach.

5.2.3 Experiment 7: Fast Network Interface

For fast network interfaces (graphs available in [6]), the performance behavior was similar to that of the previous two experiments except that a bigger gap opens up in the Pure DC case between EP and PROMPT. This is because the reduction in overheads of EP has lesser impact due to the improved communication infrastructure enjoyed by all protocols. Further, PROMPT also comes closer to PEP in performance, but is still visibly worse over the entire loading range.

5.2.4 Experiment 8: Highly Distributed Transactions

The results for higher degree of distribution are shown in Figures 5a and 5b for the RC+DC and Pure DC environments, respectively. We see in these figures, for the first time, PEP doing *worse* than PROMPT, although still significantly better than EP. In Figure 5a, PEP is marginally worse than PROMPT, whereas in Figure 5b, it is substantially worse.

The reason for the above behavior is the following: The higher degree of distribution makes the priority inversion periods much longer, as testified to by the significant gap between the performance of EP and PROMPT in both the figures. Here, 2PL-HP moves much quicker towards becoming unprioritized 2PL since each transaction is now composed of a long sequence of short cohorts. This was confirmed from the restart statistics output by our simulator – EP had virtually no restarts at all in this environment, whereas in PROMPT the restart ratio is much higher. In the Pure DC case, it results in the EP system effectively running *in a non-real-time mode*, that is, without prioritized concurrency control. While PEP significantly reduces this problem due to its lending mechanism, the improvement is not sufficient to completely obliterate the gap between EP and PROMPT.

While the above results do highlight a limitation of EP and PEP, in practice, however, it may not be a serious handicap. This is because transactions are usually distributed to a limited degree, often involving only two sites [10].

6. Conclusions

Motivated by the need to reduce commit processing overheads, namely overheads due to communication and forced disk writes, we investigated here the use of 1PC protocols for distributed real-time databases. In 1PC, the voting phase of the multi-phase commit protocols is eliminated by having the cohorts of each distributed transaction enter

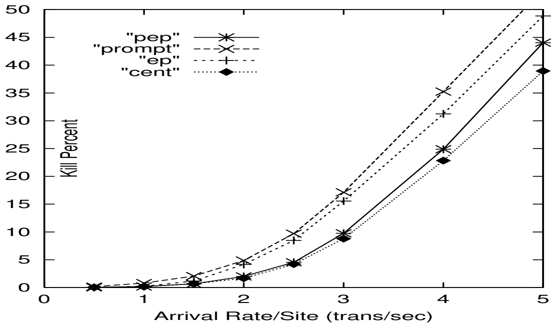

Figure 2a: Parallel Execution (RC + DC)

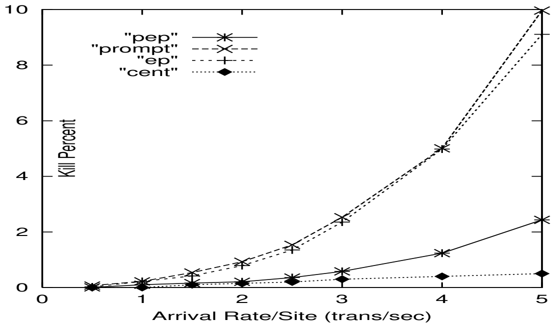

Figure 2b: Parallel Execution (Pure DC)

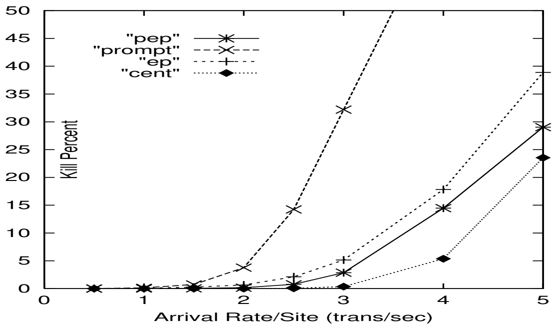

Figure 3a: Highly Distributed (RC+DC)

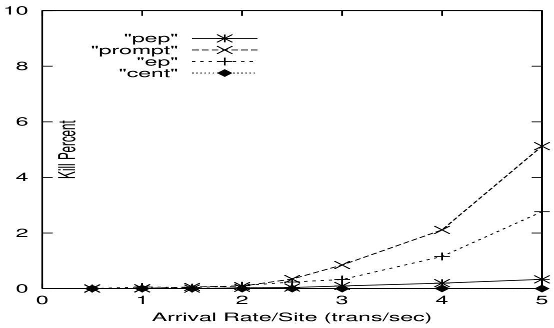

Figure 3b: Highly Distributed (Pure DC)

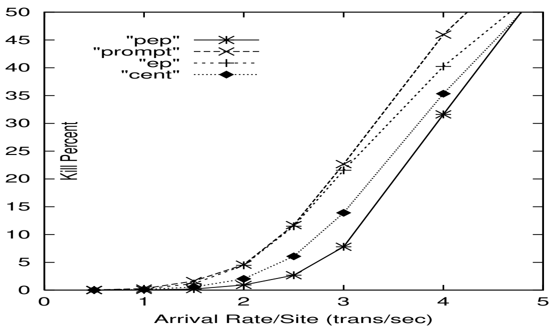

Figure 4a: Sequential Execution (RC + DC)

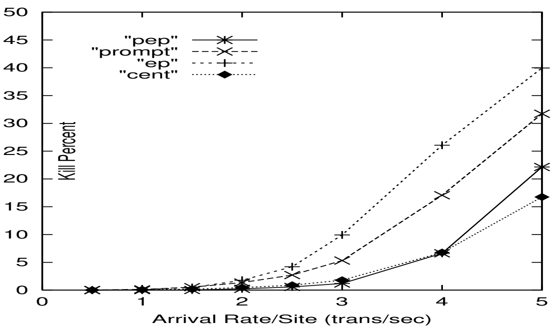

Figure 4b: Sequential Execution (Pure DC)

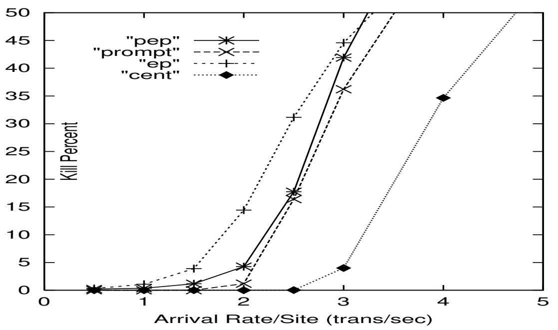

Figure 5a: Highly Distributed (RC+DC)

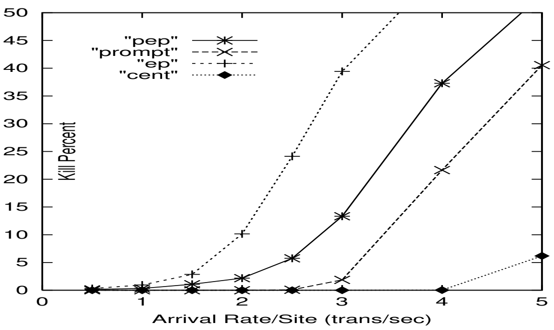

Figure 5b: Highly Distributed (Pure DC)

Table 5. Summary of Experimental Results

Execution	Network	Contention	Distribution	PEP	EP	PROMPT
Parallel	Slow	–	–	Good	OK	Bad
Parallel	Fast	RC+DC	–	Good	OK	Bad
Parallel	Fast	DC	–	Good	Bad	OK
Sequential	–	RC+DC	Low	Good	OK	Bad
Sequential	–	DC	Low	Good	Bad	OK
Sequential	–	–	High	OK	Bad	Good

the prepared state as soon as they send the work completion message to their master. But, in 1PC protocols, cohorts are in the prepared phase for an extended period of time, during which their data locks cannot be pre-empted. We addressed this issue by adapting the technique of controlled borrowing of prepared data, which we had previously proposed and successfully utilized for multi-phase commit protocols. The resulting PEP protocol obtains many of the advantages of one-phase operation and controlled sharing, as exemplified by the EP and PROMPT protocols, respectively.

Table 5 summarizes the relative performance of the protocols, synthesized from our simulation experiments. Here the comparison is relative with "Good" and "Bad" being two ends of the performance spectrum, and "OK" lying in between. If the entry for a particular system characteristic is marked by a "–" it denotes that the relative performance of the protocol is not affected by that characteristic. What is appealing about the table is that PEP is good in all but one situation: when the cohorts execute sequentially and are spread across a large number of sites. But as mentioned earlier, this situation occurs only rarely in practice. Hence, it appears safe to say that PEP is preferable in a majority of situations that occur in time-critical database applications.

Further, the above performance was derived using a very conservative implementation of PEP. In our current work, we are investigating the improvements that can be obtained from more sophisticated choices for the lending policy, the deadlock resolution mechanism and the Presumed Commit implementation.

Acknowledgments

The simulator used in our study is largely based on one developed by Ramesh Gupta, which was augmented by Prabodh Saha and Sorabh Doshi to include the one-phase protocols. The work of Jayant Haritsa was supported in part by research grants from the Dept. of Science and Technology and the Dept. of Bio-technology, Govt. of India. The work of Krithi Ramamritham was supported in part by the National Science Foundation of the United States under grant IRI-9619588.

References

[1] R. Abbott and H. Garcia-Molina, "Scheduling Real-Time Transactions: a Performance Evaluation", *Proc. of 14th Intl. Conf. on Very Large Databases*, August 1988.

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

[3] J. Gray, "Notes on database operating systems", *Operating Systems: An Advanced Course, Lecture Notes in Computer Science*, 60, Springer-Verlag, Berlin, 1978.

[4] J. Haritsa, M. Carey, and M. Livny, "Data Access Scheduling in Firm Real-Time Database Systems", *Real-Time Systems Journal*, 4 (3), 1992.

[5] J. Haritsa, K. Ramamritham, and R. Gupta, "The PROMPT Real-Time Commit Protocol", *IEEE Trans. on Parallel and Distributed Systems*, 11(2), February 2000.

[6] J. Haritsa and K. Ramamritham, "Adding PEP to Real-Time Distributed Commit Processing", *Tech. Report 2000-03*, http://dsl.serc.iisc.ernet.in/pub/TR/TR-2000-03.ps.

[7] J. Huang, J. Stankovic, K. Ramamritham, D. Towsley and B. Purimetla, "On Using Priority Inheritance in Real-Time Databases", *Real-Time Systems Journal*, 4 (3), 1992.

[8] B. Lampson and D. Lomet, "A New Presumed Commit Optimization for Two Phase Commit", *Proc. of 14th Intl. Conf. on Very Large Databases*, August 1993.

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

[10] 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.

[11] L. Sha, R. Rajkumar, and J. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization", *Tech. Rep. CMU-CS-87-181*, Dept. of Computer Science, Carnegie Mellon Univ., 1987.

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

[13] J. Stamos and F. Cristian, "A low-cost atomic commit protocol", *Proc. of 9th IEEE Symp. on Reliable Distributed Systems*, October 1990.

[14] D. Tygar, "Atomicity in Electronic Commerce", *Internet Besieged*, Addison-Wesley and ACM Press, October 1997.