Replication, Consistency, and Practicality: Are These Mutually Exclusive?

Todd Anderson

Department of Computer Science
University of Kentucky
Lexington, KY 40506
anderson@dcs.uky.edu

Yuri Breitbart
Henry F. Korth
Avishai Wool

Bell Laboratories, Lucent Technologies Inc.
600 Mountain Avenue
Murray Hill, NJ 07974
{yuri,hfk,yash}@research.bell-labs.com

Abstract

Previous papers have postulated that traditional schemes for the management of replicated data are doomed to failure in practice due to a quartic (or worse) explosion in the probability of deadlocks. In this paper, we present results of a simulation study for three recently introduced protocols that guarantee global serializability and transaction atomicity without resorting to the two-phase commit protocol. The protocols analyzed in this paper include a global locking protocol [10], a "pessimistic" protocol based on a *replication graph* [5], and an "optimistic" protocol based on a replication graph [7]. The results of the study show a wide range of practical applicability for the *lazy* replica-update approach employed in these protocols. We show that under reasonable contention conditions and sufficiently high transaction rate, both replication-graph-based protocols outperform the global locking protocol. The distinctions among the protocols in terms of performance are significant. For example, an offered load where 70% - 80% of transactions under the global locking protocol were aborted, only 10% of transactions were aborted under the protocols based on the replication graph. The results of the study suggest that protocols based on a replication graph offer practical techniques for replica management. However, it also shows that performance deteriorates rapidly and dramatically when transaction throughput reaches a saturation point.

1 Introduction

The management of replicated data in a distributed database is an old problem of great practical importance. Distributed data warehouses and data marts contain very large replicated databases distributed among a number of sites. Depending upon the application, the number of sites may range from only a few sites to several hundred sites. Telecommunication applications require rapid distribution of updates to all replicas with strong guarantees of consistency and availability.

Early work on replication, (see, e.g. [14] for a survey) focused on *eager* replica update propagation. Under eager propagation, the writing of updates to all replicas is part of a single transaction. This implies that the transaction size – the number of database operations in the transaction – grows with the degree of replication. Since the deadlock probability grows as the fourth power of the transaction size [11], the eager approach does not scale up well to large databases with a high degree of replication. Furthermore, it does not appear that straightforward modifications of eager replica update propagation can eliminate these deficiencies.

Recently, much attention has been directed towards the *lazy* approach to replica update propagation. The lazy approach requires that installation of updates to replicas occur only after the update transaction has committed at the origination site [8, 10, 13, 16, 17, 5, 1]. Propagation is performed by independent subtransactions spawned only after the transaction at the origination site has committed. Clearly, the activities of these subtransactions must be managed carefully to ensure global consistency and transaction atomicity.

In [8] the authors introduced the data-placement graph, and proved that replica consistency can be guaranteed by ensuring the acyclicity of that graph. However, their approach does not guarantee global serializability in the general case. It requires that each local DBMS use rigorous two-phase locking [4]. Furthermore, since a site normally contains a large database and the number of sites is usually much smaller than the number of data items, the data-placement graph becomes cyclic quite rapidly, which adversely affects transaction throughput. In [17] another approach to the management of replicated data was described. It ensures eventual replica convergence but does not guarantee global serializability. An alternative approach to replication appears in [1]. In [5], we proposed a new approach for guaranteeing global serializability. The protocol reduces the probability of distributed deadlock and lowers the communication overhead as compared with prior work, including [10]. We are not aware, however, of any performance studies either comparing the lazy replica update protocols or analyzing the performance of any specific lazy replica update protocol.

The purpose of this paper is to fill the gap by reporting the results of a simulation study comparing the performance of three replication management protocols that are based on lazy replica update. The first protocol we tested was first outlined in [10] and a more precise version (with a correctness proof) appears in [6]. The protocol uses a version of two-phase locking to synchronize read/write (rw and wr) conflicts, and the Thomas Write Rule [2] to synchronize write/write (ww) conflicts. The second and third protocols

ACM SIGMOD'98, Seattle, June 1998

we tested are based on variations of the replication graph technique described in [5]. The difference between these last two protocols is that one of them uses a pessimistic approach while the other takes an optimistic approach.

Our results show that the replication graph protocols outperform the locking protocol under reasonable contention assumptions, and that replication graph protocols do have practical applicability. The results validate the conjecture of [10] that global locking generates a significant number of deadlocks. However for all three protocols, performance deteriorates rapidly when the transaction rate leads to queuing for data or for physical resources.

In Section 2 we discuss the three protocols whose performance we are evaluating here. Section 3 describes our simulation model. Section 4 describes the results of our experiments, identifies the ranges over which the protocols are applicable in practice, and compares their relative performance. Section 5 presents a summary of our results along with some conclusions.

2 Lazy-Replication Protocols

In this section, we present the three protocols that we study. After defining the system model, we present the details of each protocol. Much of this section is an extension of the model described in [5]. The definition of virtual sites has an important change from [5] that allows the pessimistic protocol in the current paper to be less restrictive than protocol GS of [5]. This improved definition of virtual sites is incorporated also into our optimistic protocol.

2.1 System Model

We begin with a review of our transaction and data models. The database consists of data distributed over a set of sites. Data may be replicated to any degree up to and including full replication of data at all sites. For each data item, there is a unique site, called the *primary site*, that is responsible for updates to the data item. The copy of a data item at the primary site is referred to as the *primary copy*, and all other copies are referred to as *secondary copies*. The local DBMSs at each site ensure the usual ACID properties [12], and generate a serializable schedule for transactions executing at the local site. The local DBMSs are responsible for managing local deadlocks.

The site at which transaction T_i is submitted is called the *origination site* of T_i. Each transaction T_i can read data only at its origination site. Transaction T_i can update data item d only if it originated at the primary site of d. This is a significant restriction since it limits the sets of data items that may be updated by a single transaction. However, many applications adhere to this restriction. Generally, any application, in which each data item has a specific "owner" adheres to this restriction[1]. When the primary copy of a data item is updated, the new value must be propagated to all secondary copies of that data item. This new value must not be installed in a secondary copy until after the update transaction has committed at its origination site (that is, the primary site for the data it updates). Despite this restriction, the physical transfer of updates from the primary site to secondary sites can commence at any time to avoid network congestion at the end of the transaction.[2]

[1]Examples include stock prices, traffic data, and certain types of sales data.

[2]The restriction that transaction t can update data item d only if t orginated at the primary site of d can be relaxed as follows. A transaction can update *any* data item at its origination site, and propagation is done only after t has committed at its origination site. This relaxation leads to somewhat different prototocls than those described here, though our preliminary results suggest that the overall performance will be similar. The full discussions of these results will appear elsewhere.

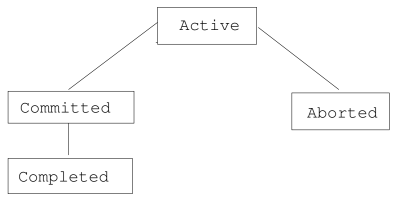

Figure 1: A general state transition diagram

Transactions that update replicated data items are referred to as *global*. All other transactions are *local*. A global transaction is represented by several local subtransactions – one for the transaction itself running at its origination site, and one for each site that holds replicas of one or more data items updated by the transaction. Consequently, the subtransactions running at remote sites on behalf of a global transaction do not begin executing until after the corresponding subtransaction at the transaction's origination site has committed.

In all the three protocols that we evaluate, a transaction T_i can be in one of the following four global states. The protocols differ in the way they decide when a state transition is to be made.

- **active**, if T_i is active at its origination site (that is, T_i has started but is neither committed nor aborted at its origination site);
- **committed**, if T_i has committed at its origination site;
- **aborted**, if T_i has aborted at its origination site;
- **completed**, if at each site at which T_i executed, T_i has committed and it is not preceded in the *local* serialization order by any transaction that has not completed.

Figure 1 depicts a state-transition diagram for the protocols. If transaction T_i is in the **active** or **aborted** state, then it has not executed any operations on replicated data items at any site except its origination site. From the **active** state, a transaction may transfer either into the **aborted** or **committed** state. Transactions cannot transfer directly from the **active** state into the **completed** state.

If a global transaction is in the **committed** state, then it may have performed some of its operations on secondary copies. From the **committed** state, a transaction can transfer only into the **completed** state. Observe that during the execution of a **committed** transaction at a site other than its origination site, the subtransaction can be aborted by the local DBMS. However, it would be restarted and reexecuted there. Generally, a **committed** transaction does not have to progress to the **completed** state, even if it has committed at all sites[5]. Our protocols, however, guarantee global serializability by ensuring that every transaction in the **committed** state eventually reaches the **completed** state. It was shown by example in [5] that it is possible for consistency to be lost unless **committed** transactions are retained in the concurrency control system until they reach the **completed** state.

Since all three protocols employ the Thomas Write Rule[3] for *ww* synchronization, we associate a timestamp with each data item. This timestamp is defined to be the timestamp of the transaction that wrote the current value of the data item.

2.2 The Locking Protocol

The locking protocol whose performance we study in this paper was first introduced in [10]. A more precise version of this protocol appears in [5]. However, the version of the locking protocol in [5] was unnecessarily restrictive. Here we introduce a less restrictive version whose performance is investigated in our experiments. We assume that each local site has a local concurrency control mechanism that guarantees the ACID properties [2]. In addition, we assume that there is a global lock manager whose functions we describe below.

In this protocol, a transaction must request a read lock from the primary site of each data item that it reads. If the transaction originates at the primary site of the data item, this lock request is directed to the local lock manager. If the transaction originates at the site of a secondary copy, the read lock request is *relayed* to the primary site. If the read lock at the primary site is granted, the read operation can proceed, otherwise the transaction is forced to wait.

As we stated earlier, each write operation on a data item must be performed first on the data item's primary copy. Consequently, if a transaction submits a write operation on a primary data item locally, the lock will not be granted until there is no other transaction reading the data item at any site. Thus, read and update operations conflict at the primary site of the data item. Until an update is completed for all replicas of data item d, no other transaction can read d. This is achieved by granting the transaction that updates the primary copy of d a lock which is not released until all data replicas have been updated. Write operations are synchronized using the Thomas Write Rule [2]. Thus, in the terminology of [2] the concurrency control mechanism uses the Thomas Write Rule to synchronize *ww* conflicts and the two-phase locking to synchronize *rw* and *wr* conflicts.

Note that as shown in [6], the locking protocol must ensure that read locks are retained until the transaction completes in order to guarantee global serializability.

2.3 Replication Graph Protocols

We begin our description of the two replication graph protocols by presenting two of their key components: the concept of a virtual site and the definition of the replication graph. Following this we describe the specific details of each protocol.

2.3.1 Virtual Sites

We divide each physical site into a dynamically changing set of *virtual sites.* Local transaction management within each virtual site is provided by the database system running at the physical site containing the virtual site. Because the protocol is part of an integrated system, we are able to use transaction management information from the local transaction managers, unlike the case for multi-database systems [3].

Each transaction has a virtual site associated with it at each physical site at which it executes. This virtual site exists from the time the transaction begins until the protocol explicitly removes it from consideration. We denote the virtual site for T_i at physical site s_j by VS_i^j. It is important to note that more than one transaction may share a virtual site. Thus, for some transactions T_i and T_k, it may be the case that VS_i^j and VS_k^j are identical (which we denote by $VS_i^j = VS_k^j$). The set of virtual sites is constructed and maintained based on the three rules below:

- **Locality rule.** We require that each local transaction execute at precisely one virtual site. Thus, a local transaction always executes at a single virtual site. A global update transaction, however, has several virtual sites – one at each physical site at which it executes. At every point in time, VS_i^j must contain the set of all data items at physical site s_j that transaction T_i has accessed[4] up to that point.

- **Union rule.** If two transactions T_i and T_k, access a data item d in common at site s_j, their virtual sites VS_i^j and VS_k^j at site s_j must be merged if one of the following hold:

  - Data item d is a primary data item and T_i and T_k conflict (directly or indirectly)[5] on d.
  - Data item d is a secondary data item and T_i and T_k are in a *rw* or *wr* conflict (directly or indirectly) on d.

  Merging of VS_i^j and VS_k^j at site s_j means that these virtual sites become the same and the merged site (now called both VS_i^j and VS_k^j) contains all the data accessed so far by T_i or by T_k at s_j.

- **Split Rule.** When physical site s_j determines that T_i has entered either the **aborted** or **completed** state, any data items accessed exclusively by T_i are removed from VS_i^j and the replication protocol need no longer consider T_i. If there is no T_k distinct from T_i such that $VS_i^j = VS_k^j$, this effectively removes VS_i^j. Otherwise, we may recompute the virtual sites at site s_j for all transactions T_k such that $VS_i^j = VS_k^j$ using the locality and union rules. This computation can be optimized using transaction conflict information to reduce overhead.

The locality rule ensures that at each physical site there is a single virtual site in which the transaction is executing. The union rule guarantees that if two transactions conflict on a data item at the transaction origination site, then these transactions execute at the same virtual site. Consequently, each virtual site can be logically considered to have its own transaction manager that does not need to handle distributed transactions over virtual sites within a physical site.

If two transactions execute write operations on the same data item, then their *ww* conflict is handled by the Thomas

[3]This rule pertains to the case where transaction t updates d but the timestamp of t is less than the write timestamp of d. Instead of aborting t, t continues as if the write succeeded, though, in fact, the write is ignored.

[4]A transaction is said to access a data item d at site s if it has executed a read of d at s or has executed a write on any replica of d regardless of site.

[5]We assume the usual notion of read and write conflict[2]. A conflict is *indirect* if it results from a series of direct conflicts.

Write Rule [2] (as we shall see in the protocol definitions). Therefore, there is no need to merge virtual sites due to a *ww* conflict. This makes it possible to keep virtual sites smaller and reduce the amount of contention during replica propagation.

Whereas the locality and union rules are requirements for correctness, the split rule is aimed at necessary performance improvements for the protocols to be practical. The power of the protocol arises from keeping virtual sites as small as possible. Thus, when a transaction T_i enters the **aborted** or **completed** states, it is desirable to use this information to split or shrink virtual sites.

2.3.2 Replication Graph

We associate a *replication graph* with an execution of a set of transactions to represent conflicts arising from updates to replicated data. There is a single, global replication graph for the entire distributed system.

Let T be a set of global transactions, T' be a set of local transactions, and V be a set of virtual sites at which transactions from $T \cup T'$ are executing. We say that $RG = < T \cup V, E >$ is a replication graph if RG is an undirected bipartite graph. For each transaction T_i in T, there is an edge (T_i, VS_i^j) in E for each physical site s_j at which T_i has a virtual site (VS_i^j). For a given schedule S over the set $T \cup T'$ we require that a replication graph be constructed in compliance with the locality and union rules; thus the replication graph evolves with time according to the schedule.

In [5, 6], it was shown that for a schedule S, if there exists an evolution of the replication graph in which the graph is acyclic at every point in time, then S is globally serializable. Thus, the key steps in the replication graph protocols (both pessimistic and optimistic) involve testing for cycles in the replication graph at certain points in the transaction's execution. Given a replication graph and a set of operations, we define a test, called **RGtest**, which tentatively applies the locality and union rules to the replication graph, for that set of operations. **RGTest** *succeeds* if no cycle results, otherwise it is said to *fail*. Typically, if **RGtest** succeeds, the set of operations is allowed and the tentative changes to the replication graph are made permanent, otherwise, the transaction involved may be required to wait or to abort. The specific details for each protocol appear in the next two sections.

2.4 The Pessimistic Protocol

In this section, we describe the *pessimistic protocol*, which is based upon protocol GS of [5]. The main distinctions between the protocols are the definition of virtual sites (see Section 2.3.1) and the use of the Thomas Write Rule to handle *ww* conflicts. These changes lead to smaller virtual sites and thus to less contention in the pessimistic protocol as compared with protocol GS of [5].

1. If T_i submits its first operation, assign T_i a timestamp.
2. If T_i submits a read or write operation at its origination site, apply **RGtest**:
   - If **RGtest** succeeds, allow the operation to execute (applying the Thomas Write Rule if the operation is a write) and make the tentative changes to the graph permanent.
   - If **RGtest** fails and T_i is local, T_i submits the abort operation.
   - If **RGtest** fails and T_i is global, test the tentative replication graph to see if any cycle includes a transaction in the **committed** state. If so, T_i submits the abort operation, else T_i waits.
3. If T_i submits a write operation at a site other than its origination site, apply the Thomas Write Rule.
4. If T_i submits the commit operation, proceed with execution. If T_i is in the **completed** state, remove it by deleting it from the replication graph (if it was present) and applying the split rule. Check whether any waiting transactions can be activated or aborted as a result.
5. If T_i submits the abort operation at its origination site, delete it from the replication graph and remove subtransactions of T_i from any waiting queues in which they appear. Apply the split rule. Check whether any waiting transactions can be activated.

Note that although local transactions do not appear as *nodes* in the replication graph, they do have virtual sites. Therefore, by the union rule, local transactions (both update and read-only transactions) affect the set of virtual sites, and can delay removal of a global transaction from the replication graph.

2.5 The Optimistic Protocol

In this section, we discuss an optimistic protocol that, like the pessimistic protocol, guarantees global serializability. Under the optimistic protocol **RGtest** is applied only once – when a transaction submits the commit operation. This results in fewer invocations of **RGtest**. Furthermore, since no waits are induced by the optimistic protocol, there are no global deadlocks possible (deadlocks purely within local DBMSs remain possible, but such deadlocks can be dealt with locally).

1. If T_i submits its first operation, assign T_i a timestamp.
2. If T_i submits a read or write operation at its origination site, the operation is executed (with the Thomas Write Rule applied if the operation is a write). In processing operations, maintain the data access set[6] of T_i for each site (this will be used in step 4 to construct appropriate virtual sites).
3. If T_i submits a write operation at a site other than its origination site, apply the Thomas Write Rule.
4. If T_i is in the **active** state and submits a commit operation (that is, T_i submits the commit operation at its origination site), apply **RGtest** to the operations of T_i. If **RGtest** succeeds, make all tentative changes permanent and perform the commit. Otherwise, cancel tentative changes and abort T_i.
5. If T_i is in the **committed** state and submits a commit operation (that is, T_i submits a commit operation at a site other than its origination site), proceed with the execution. If this results in T_i entering the **completed** state, remove it from the replication graph and apply the split rule.
6. If T_i submits the abort operation at its origination site, proceed with the execution. Apply the split rule.

[6]Note that operations ignored under the Thomas Write Rule still count as part of the access set.

The optimistic protocol allows each transaction to proceed at its origination site independently of other transactions that are executing at other sites. The only coordination required is when the transaction submits the commit operation at its origination site.

2.6 Performance Expectations

Prior to beginning our performance studies, we expected that the locking protocol would perform best in low-contention scenarios because it does not incur the overhead of replication graph maintenance. For higher contention, we anticipated that the replication graph protocols would outperform the locking protocol. It was not clear to us which of the replication graph protocols would perform best. It was arguable that the pessimistic protocol should perform better for sufficiently high contention since it tends to abort transactions earlier. The optimistic protocol, by deferring all graph testing until the transaction commits at its origination site, could extend the life of transactions destined to abort. This results in increased contention within the local DBMSs and could cause the optimistic protocol to underperform the pessimistic protocol. Conversely, the optimistic protocol requires less access to the centrally maintained replication graph, and thus promises reduced load on a part of the system that appears likely to be a bottleneck.

3 The Simulator

To evaluate the performance of the three protocols, we developed a simulation model in C++ using the simulation package CSIM [9]. The simulated system consists of a set of sites connected by an ATM network. One of these sites serves as the replication graph manager, and is called the *graph site*. The other sites contain parts of the database and run a simulated local DBMS. We refer to these latter sites as *database sites*. All three protocols were tested with the same number of active database sites.

Our designation of a site to serve exclusively as the graph site was a matter of implementation convenience. In practice, one of the database sites could serve in this role. By separating this function, we were able to distinguish between graph-induced load and database-induced load, both in terms of network traffic and CPU load in managing the replication graph. Clearly, the central graph site eventually becomes a bottleneck as the number of sites and the number of transactions grow. However, we wanted to determine where that point occurs. From our experiments, we found that significant transaction loads could be processed without any bottleneck developing at the graph site, particularly for the optimistic protocol. The locking protocol did not make any use of this extra graph site.

The database sites are modeled using several components: a single CPU, one or more disks, a connection to the ATM network, a two-phase locking transaction manager (to ensure local serializability), a transaction generator, and one or more transactions. Separate threads are used for the network, the lock manager, the transaction generator, as well as for each transaction. The network thread manages data transfer over the simulated ATM network. The lock-manager thread waits for transactions to send requests to it and then responds to those requests.

A central parameter to the simulator is the global submitted transaction rate, measured in transactions-per-second, and denoted by *TPS*. Each database site generates an equal portion of the global *TPS*, i.e., $locTPS = TPS/\#sites$. The transaction generator starts transaction threads according to an exponential arrival distribution with a mean determined by *locTPS*. Individual transaction threads are responsible for determining the characteristics of each transaction (read-only/update, number of operations, etc.) and for generating the transaction's operations. Transaction operations are first sent to the local lock manager. Once the lock manager responds, the transaction thread processes the operations in accordance with the protocol being employed (locking, pessimistic or optimistic).

Deadlocks are managed by a timeout mechanism rather than by global deadlock detection. The timeout interval is a parameter to the simulator, but our experiments with changing this parameter showed relatively little sensitivity.

We made a worst-case assumption that all data is replicated at all sites. We assumed a sufficiently large number of disks and sufficiently high hit ratio for the database buffer in main memory to ensure that disk access would not limit performance. Specifically, we simulated 10 Seagate Barracuda-9 disks per site and a 90% hit ratio on the buffer.

We needed to have a degree of contention for data among transactions so as to show clearly any performance differences among the protocols, since almost any protocol performs well in the absence of contention. Furthermore, a low level of contention would have necessitated extremely long simulation runs in order to obtain statistically significant results. For these reasons, we chose to simulate only the "hot spots" of the database – that is, the part of the database that gets a disproportionate share of the accesses. We chose a relatively small data item size (1KB) so that the overhead of shipping data items would not dominate the performance distinctions between protocols.

We chose a CPU speed of 300 MIPS, reflecting the speed of recently announced processors.

The network is modeled as a star with an ATM switch at its center. Each site has two links with the switch: An incoming link and an outgoing link. When a site sends a packet, it uses its outgoing link to send the packet to the switch. The switch delays the packet by a length of time corresponding to the latency of the network and then uses the incoming link of the destination machine. The network is multicast and broadcast capable. In each multicast or broadcast message, the outgoing link of the sending machine is used only once for the message but every recipient machine's incoming link is used when that message is received. Within this basic network framework, we considered both a metropolitan-area network (which is referred to as an OC-3 ATM network) and a network of continental scope (which is referred to as an OC-1 ATM network).

The simulator included a complete implementation of all the data structures and code for maintaining the replication graph. Therefore the costs we associated with operations on the replication graph are based on compiling the code into assembly language and counting the numbers of instructions that were generated.

In practice each protocol would need to include a mechanism for fault tolerance. We did not simulate any specific mechanisms here, on the grounds that similar techniques can be applied regardless of the choice of protocol.

Table 1 lists the parameters to the simulator and the ranges of values that we considered in our study.

4 Experimental Results

In choosing experiments to run under the simulator, we sought first to determine the maximum sustainable transac-

Parameters	OC-3	OC-1	OC-1*	vsN
General parameters				
Database sites (*#sites*)	100	100	20	*2–140
Timeout interval	0.5 sec			
CPU speed	300 MIPS			
Transaction parameters				
Read-only transactions	90%			
Update transactions	10%			
Writes in an update transaction	30%			
Operations per transaction (*#ops*)	5–15 (10 average)			
Global transactions per second (*TPS*)	*200–2600	*200–2400	*100–2400	*30–2100
Local transactions per second (*locTPS*)	*2–26	*2–24	*5–120	15
Data item parameters				
Probability that a data item is replicated	100%			
Degree of replication for replicated items	100%			
Data item size	1KB			
Primary items per site (*IPS*)	20			
Total number of items ($\lvert DB \rvert$)	2000	2000	400	*40–2800
Network parameters				
Latency	0.004 sec	0.1 sec	0.1 sec	0.1 sec
Bandwidth	155 Mb/sec	55 Mb/sec	55 Mb/sec	55 Mb/sec
Disk parameters				
Latency (Seagate Barracuda 9)	0.0097 sec			
Transfer rate (16 bit UltraSCSI)	40MB/sec			
Disks per machine	10			
Buffer miss ratio	10%			
Replication graph parameters				
Cost to add operation to graph	2000 instructions			
Cost to check node during cycle checking	117 * number of edges			
Queue bound at graph site	300			

Table 1: Simulation model parameters for the reported studies. Value ranges prefixed by a "*" indicate parameters that were varied in each study.

tion processing rate (TPS) for a variety of parameter values. This allowed us to determine the range of applicability of the protocols and compare their performance. Furthermore, we sought to understand the nature of the bottlenecks that ultimately limit the transaction processing rate. We studied abort rates, load of various system components (disk, CPU, network), queuing and CPU load at the graph site (for the pessimistic and optimistic protocols), and degree of data contention. Since for many applications, response time is as important as overall throughput, we also examined the time between entry into the **committed** state and entry into the **completed** state. This latter metric provides a good measure of the response-time overhead imposed by the protocols. We set database size to be relatively small so as to represent only heavily accessed data contained in the "hot spots" of a practical application. Had we considered a full database rather than just "hot spots," we would have reported somewhat higher absolute transaction rates (transactions per second), but also would have faced greater difficulty obtaining statistically significant results. The overall transaction rate is limited also by our assumption of replication of all data items at all sites. Under this assumption, each update transaction leads to a significant degree of contention. As a result, our experiments all show a low throughput rate for update transactions although the rate for read-only transactions is high. Had we bounded the degree of replication at a low value (e.g., 5), the number of transactions per second per site would be much higher, especially in our 100-site experiments.

Each point in our graphs represents the output of run of 100,000 transactions. Transients[15] were eliminated by discarding the first 5 transactions generated by every site. The final measurements were taken when the 100,000'th transaction was *submitted*, so as to avoid "system wind-down" effects. For such runs, the graphical height of the 95%-confidence intervals[7] was very close to the size of the symbols depicted on the curves. Therefore in order to reduce the complexity of our graphs, we do not show these intervals in all but Figure 7 (where the variability is large enough to merit them). However, all the distinctions we address in our discussions are indeed statistically significant differences.

4.1 OC-3

We begin by considering an OC-3 metropolitan-area ATM network example (see Table 1).

4.1.1 The Completion Rate

In our experiment, we varied the rate of submitted transactions (*TPS*) until we reached the point where the protocol in question began to behave poorly. Figure 2 shows the transaction completion rate as a function of submitted load. At low loads, contention among transactions is virtually nonexistent and the curves for all the protocols have a slope of nearly 1 (i.e., almost every transaction completes). At increasing loads, the slope approaches 0, and for very high loads, the slope turns negative. We are not interested in performance past the saturation point (since a "gate-keeper" could be used to limit the submitted load), but we are interested in how abruptly a protocol's behavior turns bad.

[7] A 95% confidence interval means that that value appears within the specified interval with probability 0.95. See [15] for a precise definition of confidence interval.

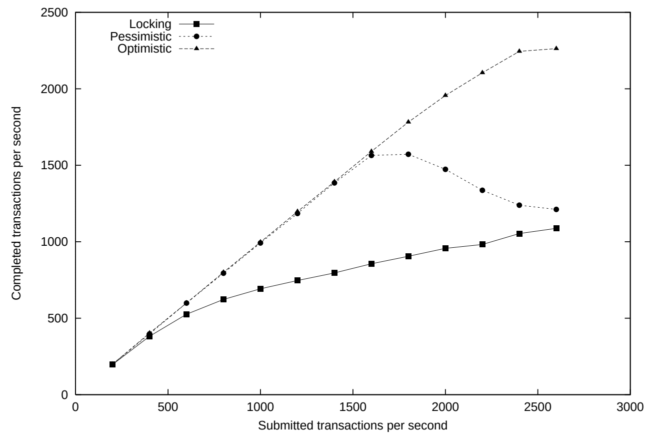

Figure 2: Number of completed transactions, OC-3 study.

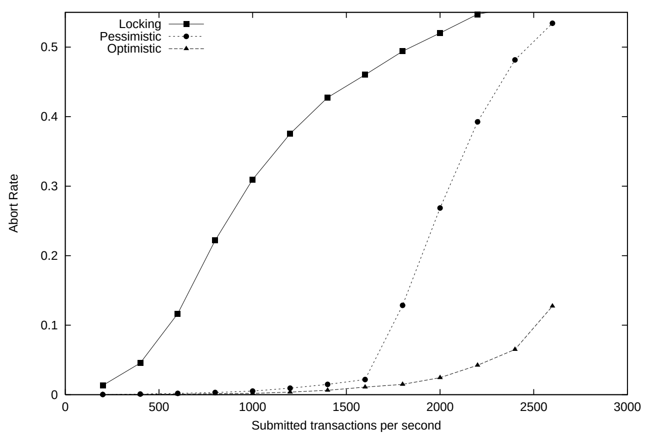

Figure 4: Fraction of transactions that were aborted, OC-3 study.

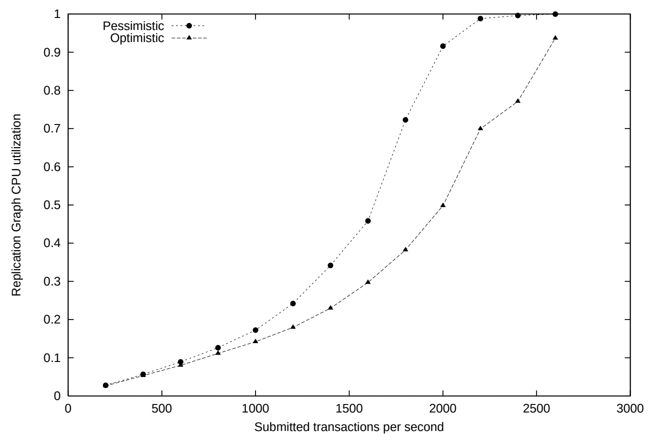

Figure 3: Graph site CPU utilization, OC-3 study.

Figure 2 shows that the optimistic protocol performs better than the pessimistic one, which, in turn, performs better than the locking protocol. The differences among the protocols increase significantly with submitted load. Despite our initial expectations of the locking protocol outperforming the replication graph protocols for low contention, we found that it only equaled the replication graph protocols at low load and fell behind quickly with increasing load.

To understand the source of contention, we plotted disk utilization and network utilization. None of these grew excessively high – the only figure of note being 55% disk utilization for the highest load – so we have not included these graphs in the paper. Figure 3 shows CPU utilization at the graph site for the two replication graph protocols (pessimistic and optimistic). The CPU utilization at the graph site rises rapidly at the load levels where the protocol's performance falters. Thus, for this set of experiments, the processing of the replication graph is the bottleneck.

While it is not surprising that the central site for graph management is a bottleneck, it is interesting to note that the source of the problem is not the number of RGtests submitted (which grows linearly in the submitted load). Rather, it is the increased complexity of graph management arising from the growth in the size of virtual sites that leads to the super-linear growth in CPU requirements. Specifically, as contention increases, virtual sites become larger and more costly to manage (under the union and split rules).

Although graph overhead is the ultimate limiting factor to the performance of the pessimistic and optimistic protocols, this limit is significantly higher than the corresponding limit for the locking protocol. Locking protocol throughput grows slowly after reaching about 600 TPS, while the replication graph protocols handle 1600 TPS (for the pessimistic protocol) to 2400 TPS (for the optimistic protocol) before performance limits are reached. The superior performance of the replication graph protocols is shown more dramatically by plotting the abort rate as a function of load (see Figure 4). The abort rate for the pessimistic protocol is negligibly small until a load of 1600 TPS. At this point, the delays in the graph site cause a significant number of transactions to be queued for processing at the graph site. When this queue grows too long (we set a limit of 300 transactions), any new entrants to the queue are aborted. The optimistic protocol continues with a low abort rate until well past 2000 TPS of offered load and suffers only a gradual increase in aborts past that point.

4.1.2 Bounding the Queue Length

Before we instituted the bound on the queue at the graph site, the replication graph protocols generated queues of excessive length, and in turn the replication graph itself grew larger. When the load approached the saturation point, the large amount of processing time for graph maintenance caused the graph site to take longer to process each **RGtest** due to CPU over-utilization. Eventually, queued transactions timed out based on the timeout interval used for deadlock management. This problem was particularly apparent in the pessimistic protocol. We observed serious instability in the performance of the pessimistic protocol at moderately high load and a dramatic deterioration in the pessimistic protocol performance at the saturation point. After a series of experiments, we settled on the technique of bounding the size of the queue at the replication graph site. Based on analysis and experimentation, we settled on a bound of

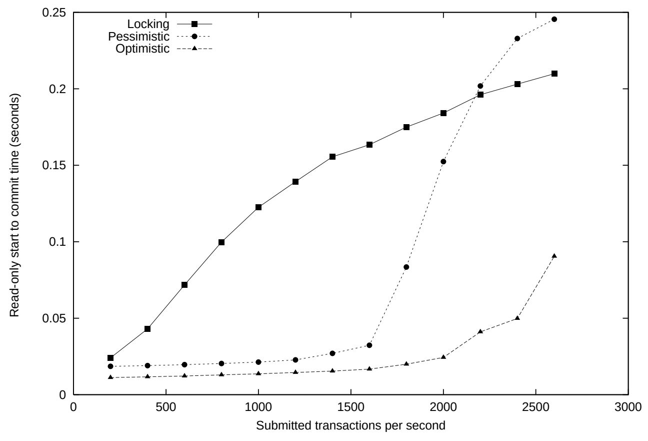

Figure 5: Response time for read-only transactions, OC-3 study.

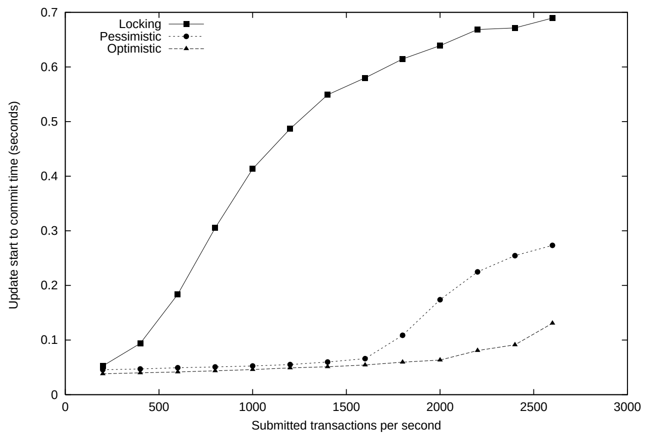

Figure 6: Response time for update transactions, OC-3 study.

300 queue entries. Further tests showed that this bound allowed the pessimistic protocol to achieve graceful degradation under load without any instability. Our experiments with various queue bounds showed that the the vast majority of aborts under the pessimistic protocol are due to the bounded queue length[8], but overall performance is not highly sensitive to the specific choice of bound.

This implementation detail, though simple, is quite important to the practicality of the pessimistic protocol. With the earlier, rapid transition to saturation, it would have been necessary in practice to limit the submitted load to well below the saturation point. Our current improved implementation allows us to run a practical system at loads much closer to the point of peak throughput.

4.1.3 Response Time

Let us now address a different performance issue: transaction response time. Figures 5 and 6, respectively, show the time from the start of a transaction to its entry into the **committed** state for read-only and update transactions. Not surprisingly, read-only transactions show better response time. More interesting is the wide disparity in response time between the locking protocol and the replication graph protocols (at least until their saturation point). Long transaction lifetimes increase the probability of waits and deadlocks under the locking and pessimistic protocols, and, thus, it is not surprising that the curve for abort rate tracks the curves for transaction duration.

Using the figure of 0.06 sec duration for an update transaction (valid up to 1600 TPS for the optimistic and pessimistic protocols), there are on average 96 concurrently executing transactions. Each transaction accesses 10 data items on average. Assuming each transaction is halfway through its updates (on average), 480 of the 2000 data items are locked for either read or write. This is a high contention level for a database system (but recall that we are actually simulating only the hot spots of the database). The fact that the replication graph protocols perform well even at this high levels of contention is a strong indication the the range of practical applicability of these protocols is high.

[8]In the OC-3 study, for example, 67% of all aborts under the pessimistic protocol at an submitted load of 1800 TPS are due to the queue being full.

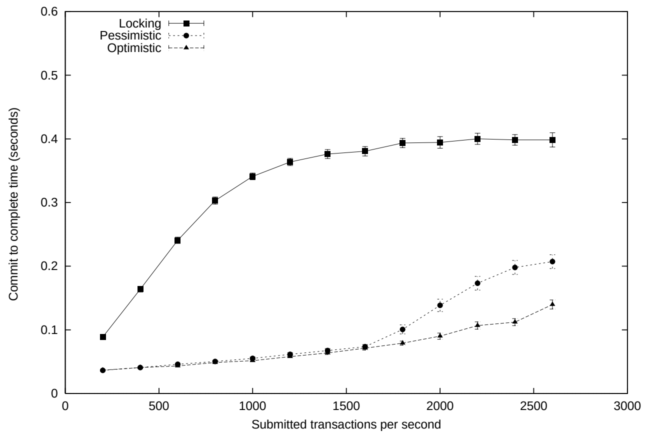

Figure 7: Time from commit to complete for update transactions, OC-3 study.

Figure 7 shows the average amount of time between a transaction's entry into the **committed** state and its completion. This measures the time taken to accomplish lazy update propagation. Once again, the replication graph protocols outperform the locking protocol. The optimistic protocol outperforms the pessimistic protocol at high load. The locking protocol times rapidly rise to just less that 0.4 seconds (that is, very close to the timeout interval of 0.5 seconds).

4.2 OC-1

We repeated our experiments for a slower network to see if network limitations would reduce the advantages shown by the replication graph protocols. We kept all parameters other than those pertaining to the network unchanged. Our parameters are based on an OC-1 continental network. Figure 8 shows the completion rate as a function of submitted

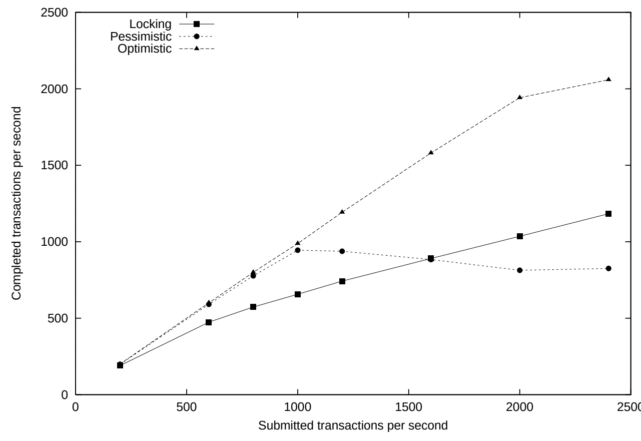

Figure 8: Number of completed transactions, OC-1 study.

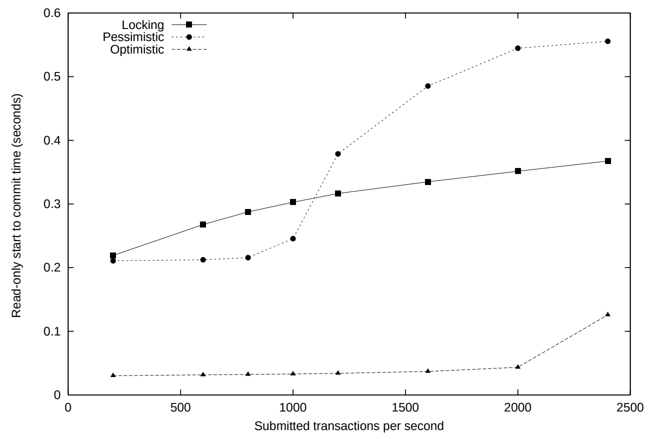

Figure 9: Response time for read-only transactions, OC-1 study.

load and should be compared with Figure 2. It is apparent that the pessimistic protocol suffers most under the slower network though it still outperforms the locking protocol for a submitted load of 1600 TPS or less. The optimistic protocol is virtually unaffected by the slower network, and outperforms the locking and pessimistic protocols even more dramatically than in the OC-3 case.

The slower network increases the overall transaction response time. The optimistic protocol is least susceptible to this increase since a transaction needs to consult the replication graph only once whereas the pessimistic and locking protocols require communication (for graph update and locks, respectively) on each operation. Figures 9 and 10 show the response times of read-only and update transactions. The replication graph protocols provide significantly better response time for update transactions. The locking and pessimistic protocols are comparable in their response time for read-only transactions (until past the 1000 TPS saturation point for the pessimistic protocol), but the optimistic protocol is better than both the locking and pessimistic protocols by factors of 7.7 and 6.1, respectively, up

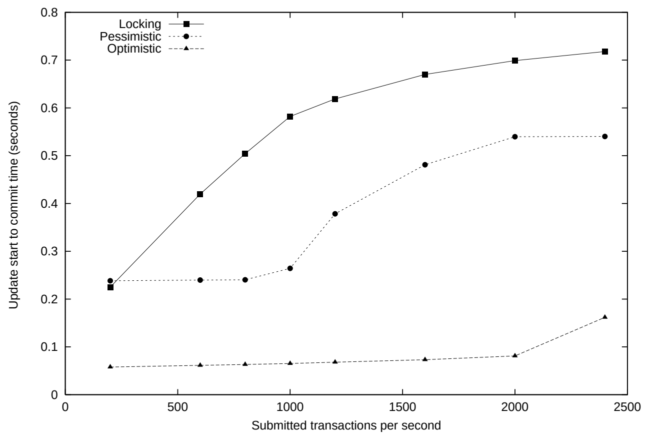

Figure 10: Response time for update transactions, OC-1 study.

to 1000 TPS, and better by factors of 8.1 and 12.6, respectively, at 2000 TPS.

Although we do not show our graph for network utilization due to space constraints, it suffices to state that overall utilization remained low (less than 10%), so latency, not traffic, is the dominant factor.

Overall, our conclusions for the slower OC-1 network are similar to those for the OC-3 network. The slower network worsens the graph-site overhead for the pessimistic protocol, but the optimistic protocol retains clear superiority.

4.3 OC-1 With a Reduced Number of Sites

Next, we consider another OC-1 scenario in which we reduced the number of sites from 100 to 20. We retained the assumption of 20 primary data items per site, thus reducing the database size to 400 data items. In Table 1, we denoted this scenario by OC-1^*.

This reduced size obviously increases the contention rate for any submitted transaction load. Figure 11 shows the completion rate as a function of submitted load for this scenario.

We continued to raise the submitted load beyond the point where throughput for the locking protocol actually began to decrease, and found that the curves for both the pessimistic and optimistic protocols retained a slope of nearly 1 (i.e., virtually all transactions completed).

The striking superiority of both replication graph protocols in this scenario is due to the way these protocols manage the high contention level and the relative change in load at the graph site when compared with the OC-1 scenario of Section 4.2.

Figures 12 and 13 show the CPU utilization at the graph site for the OC-1 and OC-1^* scenarios, respectively. From these graphs, we can see that by reducing the number of sites and number of data items, not only the absolute load at the graph site, but also the growth rate of that load changes significantly. Whereas in the OC-1 scenario, the utilization grows super-linearly when submitted load exceeds 600 TPS (for the pessimistic protocol) or 1000 TPS (for the optimistic protocol), it remains essentially linear in the OC-1^* scenario. Two factors contribute to this reduced load at the graph site. First, the reduction in the number of sites reduces

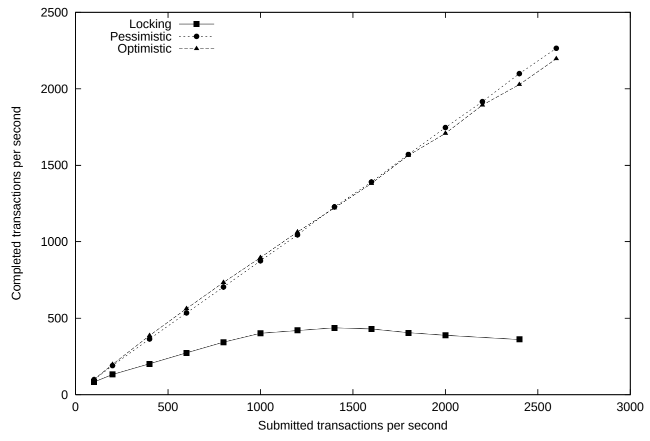

Figure 11: Number of completed transactions, OC-1* study.

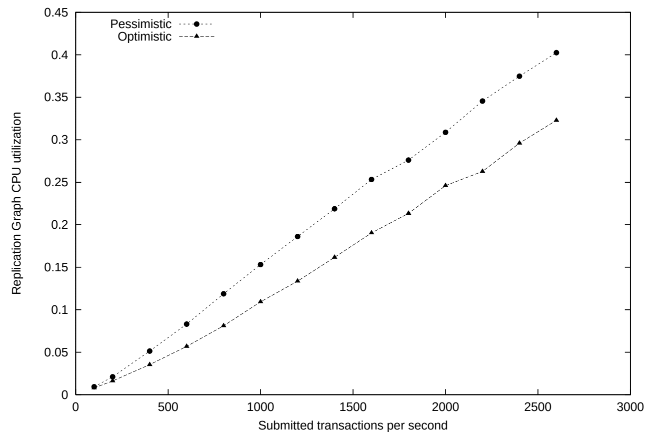

Figure 13: Graph site CPU utilization for OC-1* experiment.

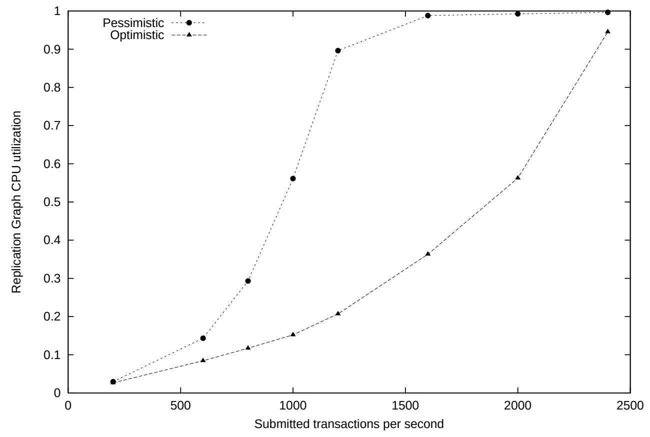

Figure 12: Graph site CPU utilization for OC-1 experiment.

the number of virtual sites per update transaction. Under our assumption of full replication to all sites, every update transaction must have a virtual site at every physical site. The second factor is that since the total number of data items in the database is reduced, the virtual sites themselves are smaller and thus, the time for processing the union and split rules is reduced.

To understand more clearly how the three protocols respond to the higher level of contention in the OC-1* scenario, consider the graph in Figure 14, showing the abort rate as a function of submitted load. It is obvious that the locking protocol has an unacceptable abort rate, with already 17% aborts at 100 TPS and 34% aborts at 200 TPS.

Much more interesting is the relative behavior of the pessimistic and optimistic protocols. Although the optimistic protocol outperforms the pessimistic protocol in terms of aborts for lower load levels, pessimistic becomes superior at approximately 1400 TPS. This result is interesting because the crossover point comes at a load level where overall throughput is continuing to increase as a function of submitted load. Furthermore, the confidence intervals of our results are such that the distinction is indeed statistically significant.

This scenario is the highest contention scenario of those we have presented so far. Because graph-site utilization stays bounded, the ability of the pessimistic protocol to detect and react to transaction conflicts earlier in their execution helps it to manage this scenario better than the other two protocols. It is interesting that locking, which is traditionally classified as having a pessimistic approach to concurrency control performs much worse than even the optimistic replication graph protocol under this scenario.

Despite the strong performance of the replication graph protocols in this scenario, the data indicates areas in which further research is needed. Our protocols treat update and read-only transactions "fairly" in the sense that neither class of transactions has priority. However, since update transactions see a greater degree of contention than read-only transactions (due to the fact that they need exclusive access and need to update secondary copies), they bear a larger share of the aborts. In certain applications, it may be desired to give update transactions priority. For example, in a stock-trading application, it is important that the current prices be posted promptly regardless of contention issues. To adapt the pessimistic and optimistic protocols to such a scenario, we need to consider a mechanism to direct a greater share of the aborts towards read-only transactions. We are currently studying several ways of achieving this, including a two-version approach, and a gatekeeper approach that bounds the number of read-only transactions submitted. We conjecture that the replication graph approach will benefit from multiple versions to a greater degree than the locking protocol.

4.4 Scaling the Number of Sites

In this section, we describe a study in which we vary the number of sites instead of directly varying the number of submitted transactions per second. We denoted this study by vsN in Table 1. Our goal was to determine how well the three protocols scale up as the distributed environment grows.

In this scenario, we keep the number of data items per site constant. Also, we assume that each site generates the same number of new transactions per second. As an ex-

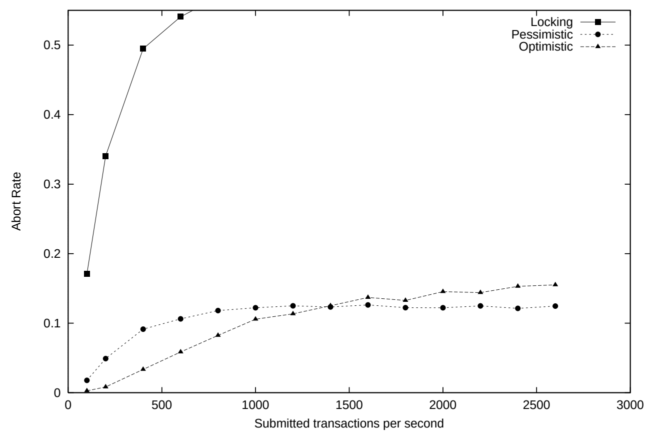

Figure 14: Fraction of transactions that were aborted, OC-1^* study.

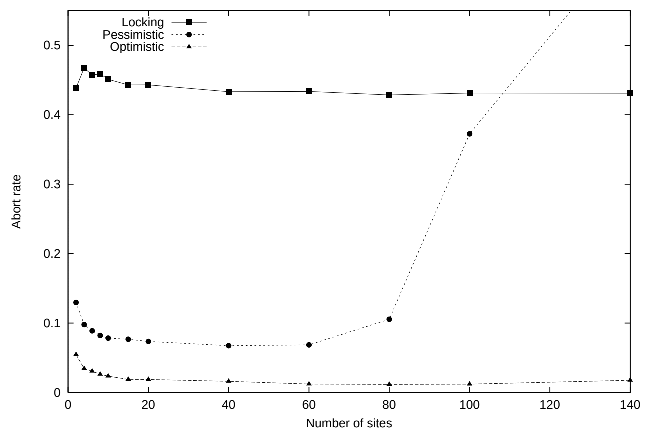

Figure 16: Fraction of transactions that were aborted, vsN study.

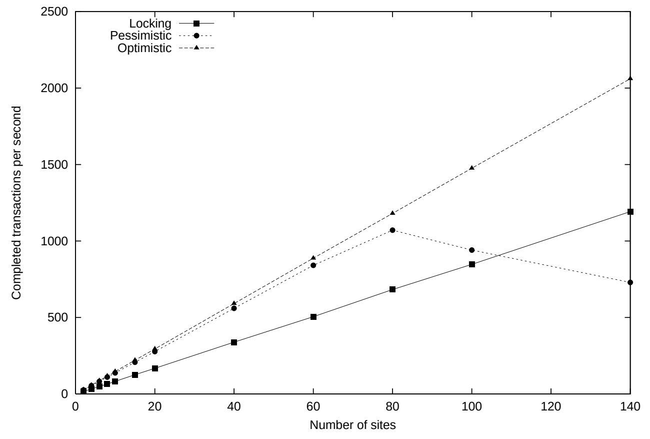

Figure 15: Number of completed transactions, vsN study.

ample, each site may represent a different stock exchange, with its own most active stocks and local traders. Thus $locTPS$ and IPS are fixed, while $TPS = \#sites \cdot locTPS$ and $|DB| = \#sites \cdot IPS$ increases as $\#sites$ grows. The ratio of transactions per second to overall database size, TPS / $|DB|$ remains constant.

In Figures 15 and 16 we present the completion and abort rates that we measured. Since each site generates a submitted load of 15 TPS, a protocol with "perfect" scaleup would have a curve with slope 15 in Figure 15. The optimistic protocol comes very close to having that level of performance. This can be seen more clearly in Figure 16, where the abort rate for the optimistic protocol is very small and is essentially identical for all system sizes (with the exception of some "edge effects" for low system sizes). The pessimistic protocol is about 5.6 times worse in abort rate as compared with the optimistic protocol but roughly 6.3 times better than the locking protocol until the system grows to 80 sites. At the 80-site level, the usual bane of the pessimistic protocol, graph-site CPU utilization, causes the throughput level to plummet. Both the locking protocol and the optimistic protocol show excellent scalability, but the optimistic protocol is able to maintain a much lower level of aborts as the system grows.

We continued our study of the effects of scaleup by considering a variant of the above scenario. In this variant, the size of the database and the global transaction rate were fixed. When the system scales up, each site owns a smaller fraction of the database and generates fewer transactions. Thus TPS and $|DB|$ are fixed, while $locTPS = TPS/\#sites$ and $IPS = |DB|/\#sites$ decreases as $\#sites$ grows. This models an environment where the system designer is free to split the database across servers based on engineering considerations such as the cost or reliability of the servers, the geography, etc. When we ran experiments under this scenario, the results were similar to those we have just presented above, thus we omit the detailed data here. Both sets of results fit well with what we expected based on analysis (see the Appendix) and lead us to predict a generally good scaleup behavior for the locking and optimistic protocols (and also for the pessimistic protocol until the graph site saturates). However, we also note that for all our experiments, the optimistic protocol continued to outperform the locking protocol by a wide margin.

5 Conclusions

We have presented the results of performance experiments for three protocols which ensure serializability in a replicated database environment. For a variety of scenarios, we have seen that the lock-based protocol we studied was outperformed by both the pessimistic and optimistic versions of our replication-graph-based protocols – and usually by a huge margin. The results clearly suggest that the system model we propose here (and in earlier work, such as [5]) is a promising one for practical applications of replicated data.

Within the domain of replication-graph based protocols, it appears that the optimistic approach is generally, though not always, better than the pessimistic approach. We identified promising directions for further study of these protocols with the goal of identifying an optimal approach in the design of replication graph based protocols.

Although the locking protocol performed poorly relative to the competition in this paper, it, too, showed a considerable range of applicability. The promising results in this paper contrast with the conjecture of [10] pertaining to unrestricted update regulation:

> This is a bleak picture, but probably accurate. Simple replication (transactional update-anywhere-anytime-anyway) cannot be made to work with global serializability.

To date, most real-world applications of replicated data avoid implementation of global serializability due to the high performance cost of global two-phase locking and two-phase commit.

Our study only begins to address issues of performance in replicated databases. We have restricted our consideration here to the case of full replication – all data items replicated at all sites. Thus, we have focused on applications of replication where the rate of read-only transactions greatly exceeds that of update transactions. For lower degrees of replication, update throughput should be significantly higher. However, the fact that the replication graph protocols have achieved a high overall throughput rate along with a significant number of concurrent updates within the overall distributed system is promising. We plan future studies to test our conjecture that the replication graph approach will show similar success in these scenarios as it did for the scenarios of this paper. Our results in this paper suggest that there is a path to global serializability without two-phase commit that achieves good performance and reduces overhead to acceptable levels.

References

[1] D. Agrawal, A. ElAbbadi, and R. Steinke. Epidemic algorithms in replicated databases. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Tucson, Arizona*, May 1997.

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

[3] Y. Breitbart, H. Garcia-Molina, and A. Silberschatz. Overview of multidatabase transaction management. *VLDB Journal*, 1(2), 1992.

[4] Y. Breitbart, D. Georgakopoulos, M. Rusinkiewicz, and A. Silberschatz. On rigorous transaction scheduling. *IEEE Transactions on Software Engineering*, 1991.

[5] Y. Breitbart and H. F. Korth. Replication and consistency: Being lazy helps sometimes. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Tucson, Arizona*, May 1997.

[6] Y. Breitbart and H. F. Korth. Replication and consistency in a distributed environment. Technical report, Bell Labs, 1997. Extended version of [5].

[7] Y. Breitbart, H. F. Korth, and A. Silberschatz. Optimistic protocols for replicated databases. Technical Report BL0112370-970227-07TM, Bell Laboratories, Lucent Technologies, 1997.

[8] P. Chundi, D. J. Rosenkrantz, and S. S. Ravi. Deferred updates and data placement in distributed databases. In *Proceedings of the Twelveth International Conference on Data Engineering, New Orleans, Louisiana*, 1996.

[9] CSIM18 simulation engine (C++ version). Mesquite Software, Inc., 3925 W. Braker Lane, Austin, TX 78759-5321.

[10] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The dangers of replication and a solution. In *Proceedings of ACM-SIGMOD 1996 International Conference on Management of Data, Montreal, Quebec*, pages 173–182, 1996.

[11] J. Gray, P. Homan, H. F. Korth, and R. Obermarck. A strawman analysis of the probability of wait and deadlock. Technical Report RJ2131, IBM San Jose Research Laboratory, 1981.

[12] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques.* Morgan-Kaufmann, San Mateo, CA, 1993.

[13] A. A. Helal, A. A. Heddaya, and B. B. Bhargava. *Replication Techniques in Distributed Systems.* Kluwer Academic Publishers, 1996.

[14] E. Holler. Multiple copy update. In *Lecture Notes in Computer Science, Distributed Systems — Architecture and Implementation: An Advanced Course.* Springer-Verlag, Berlin, 1981.

[15] R. Jain. *The Art of Computer Systems Performance Analysis.* John Wiley & Sons, 1991.

[16] C. Pu and A. Leff. Replica control in distributed systems: An asynchronous approach. In *Proceedings of ACM-SIGMOD 1991 International Conference on Management of Data, Denver, Colorado*, pages 377–386, May 1991.

[17] J. Sidell, P. M. Aoki, S. Barr, A. Sah, C. Staelin, M. Stonebraker, and A. Yu. Data replication in Mariposa. In *Proceedings of the Twelveth International Conference on Data Engineering, New Orleans, Louisiana*, 1996.

Appendix

In this section we present an analysis of the contention encountered by the protocols. Under a number of simplifying assumptions, we demonstrate that the contention is proportional to the ratio $\frac{TPS}{|DB|}$ of the global transaction rate over the total size of the database.

The strongest assumption we make in the analysis is that the *lifetime of a transaction at its originating site is independent of the contention*, where lifetime is the time until the transaction either commits or aborts. This simplification lets us avoid dealing with recursive definitions. We also make the following mild assumption: Each transaction consists of *exactly #ops* operations, on *#ops distinct* data items. As before, we only consider the fully symmetric case where each site owns the same number of items (*IPS*) and generates the same number of transactions per second (*locTPS*), and each item is fully replicated at all the sites.

We use the following notation: The lifetime of an update or a read-only transaction is denoted by ℓ_u or ℓ_r, respectively; the probability for a transaction to be an update is p_u; in each transaction, the probability for an operation to be a write is p_{wr}. The formal definition of contention we use is

Definition 1 *The* expected contention $E[C]$ *is the expected number of conflicts that a transaction participates in at its originating site before committing or aborting.*

Theorem 1 *Under the above assumptions*

$$E[C] = \alpha \cdot \frac{TPS}{|DB|},$$

where $\alpha = p_u \cdot p_{wr} \cdot \#ops^2 \cdot \Big[(1 + p_u - p_u \cdot p_{wr})\ell_u + (1 - p_u)\ell_r\Big].$

This agrees well with the approximate derivation of the probability Pr(*wait*) of a transaction having to wait in [12, eq. (7.4)]. However the expected number of conflicts allows a more precise analysis, and also gives more information. When the contention is high, Pr(*wait*) may be close to 1, yet there is a big difference if $E[C] = 1.5$ or $E[C] = 10$.