Network Latency Optimizations in Distributed Database Systems

Sujata Banerjee
Dept. of Info. Sci. & Telecommunications
University of Pittsburgh
Pittsburgh, PA 15260

Panos K. Chrysanthis
Dept. of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260

Abstract

The advent of high-speed networks will enable the deployment of data-server systems currently used in local-area networks, over wide-area networks. The users of these systems will have the same high expectations with respect to performance parameters such as the transaction throughput, response time and system reliability as in the case of local-area networks. Thus, it is important to study the performance of existing distributed database protocols in the new networking environment, identify the performance bottlenecks and develop protocols that are capable of taking advantage of the high speed networking technology. As a first step, in this paper we examine the scalability of the server-based two-phase locking (s-2PL) protocol, and discuss three optimizations which allow the s-2PL protocol to be tailored for high-speed wide-area network environments where the size of the message is less of a concern than the number of rounds of message passing. These optimizations, collectively called the group two-phase locking (g-2PL) protocol, reduce the number of rounds of message passing by grouping lock grants, client-end caching and data migration. In a simulation study, 20-25% improvement in the response time of the g-2PL protocol over that of the s-2PL protocol was observed.

1 Introduction

Several exciting advances are being made in the general area of high speed distributed computing. Network and processor speeds are increasing. Also, user desktops are being enhanced to the point that servers and clients may be completely indistinguishable in the future, with regards to computing power and functionality. The combination of faster and cheaper computers, cheaper stable memory, high-speed networks and network-aware applications is causing unprecedented growth in distributed information/data-server systems over wide area networks. The World Wide Web (WWW) is a prime example of a data-server system which is primarily read-only presently. In the future, it is expected that general data-server systems (also called *data shipping* or *enhanced client-server* systems) [1–7] in which clients perform much of their query and transaction processing locally, will be deployed over wide-area networks (WANs). We also believe that the WWW as well will evolve to require transactional support for some types of data access [8] rather than acting only as an interface to database systems [9, 10].

The initial data server systems were deployed over local area networks (LANs), owing to the low latency and relatively high speed of LANs as compared to previous generation WANs, primarily to support object oriented databases [7, 11–13]. In these systems, when a client needs a data item, it sends a request to the data-server which responds with the requested data item. Three families of caching algorithm have been proposed to preserve data consistency in the presence of concurrent requests, all derived from the widely used *strict two-phase locking* protocol (2PL) [14], namely, *Server-based 2PL, Optimistic 2PL* and *Callback Locking* [1, 2, 12, 13, 15, 16]. When data-servers become available over WANs, the users of these systems will have the same high expectations with respect to performance parameters such as the transaction throughput, transaction response time, system reliability and data availability as in the case of LANs. But as elaborated in this paper, some significant differences between LANs and WANs remain, which will necessitate modifications and optimizations to existing database protocols to obtain scalable performance to satisfy users adequately.

Given that the basic server-based 2PL (s-2PL) protocol has been found to have the best performance in situations with high data contention in LANs [5], in this paper, we study its scalability and propose three types of optimizations, namely, *lock grouping*, *deadlock avoidance* and *multiple-reads single-write*, which allow server-based 2PL protocols to be tailored for high-speed wide-area network environments in which the size of the message is less of a concern than the number of sequential phases or rounds of message passing. These optimizations enhance the performance of the database system by exploiting these characteristics of gigabit networks and in particular, they reduce the number of rounds of message passing by grouping the lock grants, client-end caching and data migration. The basic lock grouping optimization combined with the deadlock avoidance and the multiple-reads single-write optimizations are collectively called the *group two-phase locking* (or g-2PL) protocol. Although we do not deal with the recovery aspects of the g-2PL protocol here, we assume that the sites follow the standard protocol adopted by the s-2PL protocol where each site uses WAL and garbage collects its log once the data are made permanent at the server [17]. A recovery framework for the g-2PL protocol was discussed in [18].

The performance of the g-2PL protocol is evaluated using simulation, and compared with the s-2PL protocol, assuming no site and communication failures. The salient results of this performance evaluation are that the g-2PL protocol exhibits better response time for hot data and outperforms the s-2PL protocol in the presence of updates in the database system. The margin of improvement in the

response time is significant at about 20-25%.

In the next section we elaborate on the characteristics of high speed networks. In Section 3, we first briefly overview the s-2PL protocol and then present the three optimizations that form the g-2PL protocol and discuss the problem of read-dependencies in the g-2PL protocol. The simulation system model is presented in Section 4, followed by the numerical results of the performance evaluation obtained by simulation in Section 5. Section 6 concludes the paper.

2 Background

Before introducing our high speed network-specific concurrency optimizations, it is important to first discuss the characteristics of the high speed WANs and the traditional low speed networks, and understand their differences.

There are two basic components of the delay involved in moving data between two computers over a communication network: the *transmission time*, i.e., the time to transfer all the data bits, and the *propagation latency*, i.e., the time the first bit takes to arrive. Further, intermediate network components in the path of the data introduce extra delays as well. We define the sum of the delays introduced by the intermediate network components and the propagation delay as the network latency. As the data rate in WANs continues to increase due to technological breakthroughs, the data transmission delay will decrease almost linearly. However, the signal propagation delay which is a function of the length of the communication link and a physical constant, the speed of light, will remain almost constant, and relative to the data transmission delay, will actually seem to increase. This is not to say that the performance of a traditional distributed algorithm will be worse in a high speed environment than in a low speed environment. However, the marginal performance improvement will decrease as the data rate continues to increase and beyond a certain data rate there will be no further improvement since traditional database algorithms are primarily designed to optimize the data transmission delay.

Clearly, the above basic characteristic of high speed WANs (referred to as a high bandwidth-delay product) has significant implications on distributed applications. At gigabit rates in a WAN, the propagation latency is the dominant component of the overall delay [19] and the only way to combat propagation latency is to hide it in innovative protocols. Thus newer database protocols need to be developed that are *distance-independent* and scalable to wide-area high-speed networks. This observation has motivated the development of the server-based 2PL optimizations presented in the next section. An early discussion can be found in [20].

3 A New Server-based 2PL Protocol: g-2PL

In this section, after a brief review of the s-2PL protocol, we present three optimizations which can be used to enhance the performance of the s-2PL protocol in data-server systems in a wide-area high-speed network.

3.1 Server-Based Two-Phase Locking Protocol

In the basic server-based two-phase locking (s-2PL) protocol, a data-server basically preserves data consistency by following the *strict two-phase locking* protocol [14] which is the de-facto industry standard. The s-2PL protocol ensures data consistency as defined by *serializability* which requires the concurrent, interleaved, execution of requests to be equivalent to some serial, non-interleaved, execution of the same requests [21, 22].

In the s-2PL protocol, each transaction goes through a *growing phase* and a *shrinking phase*. During the growing phase, a transaction requests data items which are shipped to it after the data-server acquires a lock on them. In the shrinking phase, all the locks are released when the transaction is either aborted or committed and all modified data items are returned to the data-server. The clients are not allowed to cache locks across transaction boundaries and a client can be viewed as executing one transaction at a time. A variation of s-2PL that allows caching of locks across transaction boundaries is called *caching 2PL* (c-2PL) protocol [1, 5, 13]. To simplify the discussion, in the rest of the paper we focus only on the s-2PL protocol but the results can be easily extended to the c-2PL protocol.

Access to some data may be done in a shared fashion, with multiple clients *reading* the data item simultaneously. However, in the interest of strict consistency, while multiple clients may read the data simultaneously, no client may write on it. Hence, locks are distinguished into read (shared) and write (exclusive) types and a client cannot acquire a write lock on a data item until the clients reading the data have released their shared locks and vice versa. If the data-server cannot acquire a lock on a data item because another transaction is holding a conflicting lock on the same data, the request is enqueued and the requesting transaction is forced to wait until the lock is released.

Assuming that a transaction needs to access n data items, the first phase of the protocol as described above will involve n requests from the client to the server and n replies from the server to the client, exchanged in minimum 2 messages if all requests are sent at the same time or maximum $2n$ messages if the requests are sent sequentially. The second phase of the s-2PL protocol will involve a single message. That is, for each transaction, in the best case, the s-2PL protocol involves three *rounds*, i.e., sequential phases of message passing corresponding to lock request, lock grant and lock release and $2n+1$ rounds in worst case.

3.2 Lock Grouping Optimization

As mentioned above, at gigabit rates in wide area networks, propagation latency is the dominant communication cost which can only be reduced by minimizing both the rounds as well as the number of messages. In other words, the performance of the s-2PL protocol can be enhanced only if either the number of messages or rounds or both, involved in the protocol are reduced. We propose to reduce both messages and rounds by applying the concept of *grouping* in a novel way, previously used in various optimizations (e.g., group commit [23, 24]), and in mutual exclusion algorithms (e.g., [25]).

Specifically, we propose to *group* the lock (data) granting and release as follows. The data-server collects the lock requests for each data item and creates a *forward list* (FL) of all the clients that have pending lock requests for that data item. When a lock becomes available, the lock is granted to the first client on the forward list and the data item is sent to the client along with the forward list. For each data item required in the shared mode by multiple (reading) clients, a copy of the data item is sent to each of the reading clients. In the general case, the data structure for the forward list for each data item will be a list with appropriate markers to

Figure 1: Example execution of the g-2PL protocol: Exclusive access

delimit the parallel shared accesses and the serial exclusive access. The write has to wait until all readers have released their read locks and sent the release to the writer. While the data items have been sent out to a group of clients, the server continues to collect requests. We define the period during which the server does not possess the lock on a data item and is collecting requests as the *collection window* for the data item[1].

When a transaction commits, the client sends the new version of the committed data items to the clients next on the respective forward lists. A copy of the forward list is also sent with each data item. If the transaction aborts, the client forwards the unchanged data to the next client. Finally, when the last client on the forward list terminates, it sends the new version of the data to the data-server with the outcome of each transaction executed on the clients on the forward list. Once a data-server receives and installs the new version of a data item in the database, it dispatches the data item to the first client on the new forward list. Note that initially at start-up time and during periods of extremely light loading, the forward-list will contain a single client.

In this optimization which we call basic *group two-phase locking* (g-2PL) protocol, the lock release message of the previous client is combined with the lock grant message of the next client, thereby eliminating one sequential message required by the s-2PL protocol. For example, assume m clients under the best case where each transaction either requests a single data item or requests multiple data items within a single message. The s-2PL protocol will require $3m$ messages and $3m$ rounds as opposed to the g-2PL protocol which will require $2m + 1$ messages and $2m + 1$ rounds. The messages in the g-2PL protocol are larger than that in the s-2PL protocol, but in a high speed network environment, the message size is not a big constraint.

As another example, consider a data-server system with three clients numbered 1–3. Assume each client has issued a transaction (say, T_1, T_2, and T_3) that exclusively access the same data item. Let us assume that each message/data transfer is accompanied with 2 units of network latency and the processing time per transaction after receiving the data item be 1 unit. Further, all three transactions arrive within the same collection window. Figure 1 depicts the execution for the g-2PL and s-2PL protocols. The total execution time for our protocol is 12 units, versus 15 units with s-2PL. This is a 20% reduction in the response time of the s-2PL protocol, which will be validated later by simulation as well.

The forward list may be created according to one of several ordering rules to improve performance further. The default rule is First-In-First-Out or sort by arrival of the request as in the s-2PL protocol. As discussed below, the second and third optimizations capture two ordering rules that attempt to reduce the number of deadlocks.

3.3 Deadlock Avoidance Optimization

Two-phase locking protocols are susceptible to deadlocks [14], and so is the g-2PL protocol. Two or more transactions are said to be in a deadlock when neither of the transactions can proceed because at least one of the locks required by each of the transactions is held by one of the other transactions. Interestingly, the g-2PL protocol, described above, is susceptible to a unique type of deadlock which is created due to *read-only* dependencies formed across different collection windows. This situation is better illustrated using an example.

Consider two transactions $t_1 : read_1(x)\ read_1(y)$ and $t_2 : read_2(y)\ read_2(x)$ both of which request data items x and y for reading in a serial manner but in opposite order. As soon as the data-server gets the requests $read_1(x)$ and $read_2(y)$ will release x to t_1 and y to t_2. Now, both transactions have one data item and will not release it until they commit or abort. Subsequently, the data-server will get the requests $read_1(y)$ and $read_2(x)$ but neither data item can be released until t_1 or t_2 either commits or aborts returning x and y back to the server respectively. This is a deadlock situation where t_1 waits for t_2 to release the read lock for y and t_2 waits for t_1 to release the read lock for x.

Although read-only dependencies can be eliminated by expanding a dispatched forward list to include new read request hence avoiding read-only dependencies between read-only transactions, this g-2PL variation is not considered in this paper due to space limitation [27].

A deadlock detection and resolution algorithm that maintains a *Wait-for* graph and checks for cycles in the graph is usually coupled with any s-2PL implementation. Deadlock avoidance algorithms that ensure linear ordering are typically pessimistic requiring *predeclaration* of locks or leading to *livelocks* and hence have been considered inappropriate for dynamic databases [21, 22]. However, in

[1] Our early experimentation with a tunable collection window size and a timeout resulted in the outcome that tuning the collection window does not produce significant performance gains [26].

the case of the g-2PL protocol, deadlocks can be avoided without being pessimistic by intelligently creating the forward lists.

Specifically, deadlocks can be avoided if in each of the forward lists, the order of the transactions is the same. Formally, the forward list for each data item can be represented by a transaction precedence graph. The transaction precedence graph is a directed graph which determines the order in which each data item will *move* from one client site to another. In order to ensure linear ordering, transaction precedence graphs need to be made consistent. That is, two transactions T_i and T_j must follow the same order $< T_i, T_j >$ or $< T_j, T_i >$ in every precedence graph involving T_i and T_j. Note that the precedence graph is consistent with the lock granting order and hence consistent with the serialization order.

Clearly, this reordering of requests does not require predeclaration and because it occurs within a collection window, the problem of starvation is not encountered. In the worst case, some transactions will be pushed towards the end of the forward list but they will have the chance to access the data. In the case that such reordering of forward lists is not possible, some transactions may have to be aborted and restarted. Repeated (cyclic) restarts can be avoided in a similar way using an aging mechanism as in deadlock detection algorithms. It should be stressed that all these reordering computations are done while the server is waiting for the data items to be returned from the clients in the previous window. Thus, these computations do not increase the transaction blocking time on a lock and in fact increase the utilization of data-server CPU while reducing the transaction response time.

3.4 Multiple Reads Single Write Access

In the above optimizations, we have considered read locks and shared access to data by multiple clients. However, in the interest of strict consistency and in accord with the s-2PL protocol, while multiple clients may read the data simultaneously, no client may write on it until the clients reading the data have released the shared lock. Actually, we can do better than this by allowing multiple readers and a single writer to execute concurrently while preserving strict consistency.

In the MR1W (*Multiple Reads Single Write*) optimization, the data-server (or a releasing client) sends a copy of a data item to each of the clients on the forward list that require the data item in the shared mode along with the forward list. At the same time, it also sends a message containing the data item and the list of the shared-mode clients to the next client C_i on the forward list that requires exclusive access. In this way, C_i is enabled to execute and update the data item concurrently with the reading clients. However, C_i cannot release its updates until it receives a *release* message from all the reading clients. As before, if there are no waiting transactions that need exclusive access, the release messages are returned to the server.

Here, it is interesting to point out that with the MR1W optimization the g-2PL protocol just described behaves similar to the two-copy version s-2PL protocol [21] which allows more concurrency than the standard s-2PL protocol.

To recapitulate, in this section we first discussed a group locking optimization and derived the basic g-2PL protocol from the server-based two-phase locking protocol. Then, we discussed two more optimization types, one that avoids deadlocks and the other that enhances shared access in the basic g-2PL protocol. A detailed simulation study of the performance of the basic g-2PL protocol combined with the deadlock avoidance and MR1W optimizations is discussed in the rest of the paper. While no data access patterns have been assumed, note that the more a certain data item is requested such as hot data items, more is the performance gain, since the grouping effect is emphasized when the forward list is longer.

4 System Model for Performance Evaluation

In order to evaluate the performance of the s-2PL and g-2PL protocols under different high-speed networking latencies, simulation models of both protocols were developed using the *C* programming language. The simulation is a discrete-event simulation using the unit-time approach to advance the simulation clock [28]. We consider a data-server database system, with a single server and multiple clients connected by a high speed network. As described earlier, the transmission delays in a high speed network can be assumed to be negligible, and the network latency consists of the signal propagation and switching delays. In this paper, we make the simplifying assumption that the network latency between any two sites (server-client, client-client) and in either direction is the same.

All clients are assumed to be identical and run transactions that have the same statistical profile. The multiprogramming level at each client is assumed to be one, i.e., at any given time, each client processes a single transaction only. Further, at the end of each transaction, it is replaced with another transaction at that client site after some idle time that is uniformly distributed between a given minimum and maximum values Each transaction accesses between 1 and N data items uniformly These data items are drawn from a pool of M data items that reside at the data server. M is purposely kept small to emulate hot data access. Each data access may be of the type read with a given read probability p_r and of the type write with a probability $p_w = 1 - p_r$. The transaction execution is *sequential*, i.e., requests for data items are generated sequentially, with each request being generated only after the previous request has been granted and some think time (for computations) has elapsed. In our model, this computation time is uniformly distributed between a given minimum and maximum values.

As mentioned in the previous section, two-phase locking protocols are deadlock-prone. In the s-2PL implementation, deadlocks are detected by computing wait-for-graphs and aborting the transactions necessary to remove the deadlocks. This is the typical implementation found in commercial systems that use the s-2PL protocol. In order to avoid the use of tunable timeouts, deadlock detection is initiated when a lock cannot be granted. In the case of g-2PL, the forward lists are reordered using transaction precedence graphs to ensure that deadlocks are prevented. Recall from Section 3 that the transaction precedence graphs capture the order of lock granting and is consistent with the serialization order. In the case that such reordering is not possible, the offending transactions are aborted. Each transaction that is aborted is replaced by another transaction. The simulation model assumes that the computation cost at the data server to reorder the forward lists as well as computing the

Number of Servers	1	Percentage of read accesses	0.00 – 1.00
Number of Clients	varying	Network Latency	100 – 1000 time units
Number of hot data items	25	Computation Time per operation	1 – 3 time units
Transaction Execution Pattern	Sequential	Idle Time between transactions	2 – 10 time units
Number of data items accessed by a transaction	1 – 5	Multiprogramming level at clients	1

Table 1: Simulation Parameters

Network Type	Latency
Single Segment Local Area Network (ss-LAN)	1
Multi-Segment Local Area Network (ms-LAN)	50
Campus Area Network (CAN)	100
Metropolitan Area Network (MAN)	250
Small Wide Area Network (s-WAN)	500
Large Wide Area Network (l-WAN)	750

Table 2: Networking Environments Simulated

wait-for-graphs is the same.

Table 1 summarizes all the experimental parameters and the corresponding range of values of the performance study. Note that time durations are specified in simulation *time units* rather than real time in *seconds*. The conversion between the two is easily achieved and realistic values can be chosen by specifying the appropriate conversion factor. However, it is important to recognize that the relative values of these parameters have been chosen correctly. Since we assume a high speed WAN environment, the network latency is significantly higher than the computation/idle times. For example, if we assume that `1 simulation time unit = 0.5 msec`, then the network latencies considered are between `50 and 500 msec`, which are realistic for wide area networks including satellite transmission links. The computation time per database operation is then between `500 and 1500` μsec. In our simulations, we emulate various high speed networking scenarios, ranging from LANs to WANs, as listed in Table 2 with the corresponding network latency values.

5 Simulation Results

In this section, the results of the simulation study are presented. The g-2PL and s-2PL simulations were run on a Sun Ultra machine with the Solaris 2.5.1 operating system. The transient phase of the simulation runs was eliminated. In each simulation run, 50000 transactions (excluding the transient phase) were generated, requiring a simulation time of upto 88 million time units (upto 3-4 hours in real time). 95% confidence intervals on the average transaction response time were calculated from 5 independent simulation runs. The relative precision of the measurements never exceeded 2% of the mean values.

It is our hypothesis that the g-2PL protocol is particularly suited to accessing hot data items. Thus we simulated cases where a small number of data items are accessed by a large number of clients. Figures 2 – 4 contain the average transaction response time plotted against the network latency, for 3 values of the read probability (p_r = 0.0, 0.6, or 1.0) in a database system with 25 hot data items, 50 clients and each transaction accessing between 1 and 5 data items (uniform access) for the g-2PL and s-2PL protocols. Obviously as the network latency is increased, the average transaction response time increases correspondingly, but the lower slope of the g-2PL curve is proof of its better

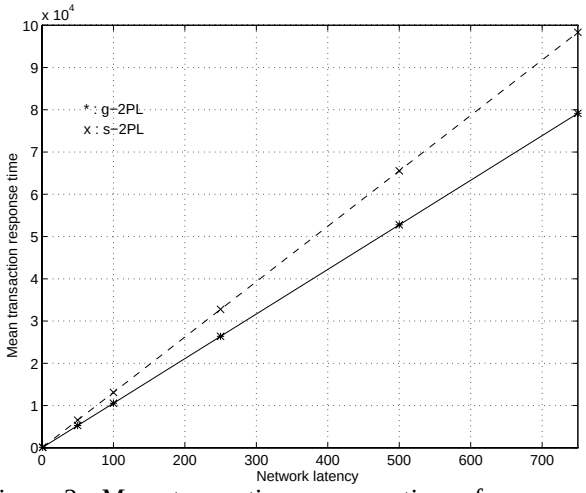

Figure 2: Mean transaction response time of g-2pl & s-2pl versus network latency, p_r=0.0

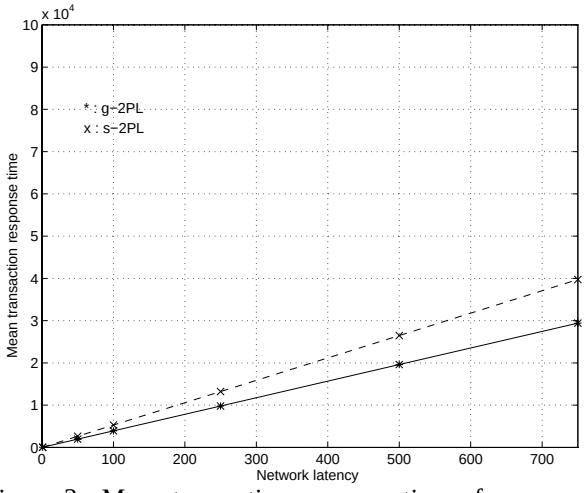

Figure 3: Mean transaction response time of g-2pl & s-2pl versus network latency, p_r=0.6

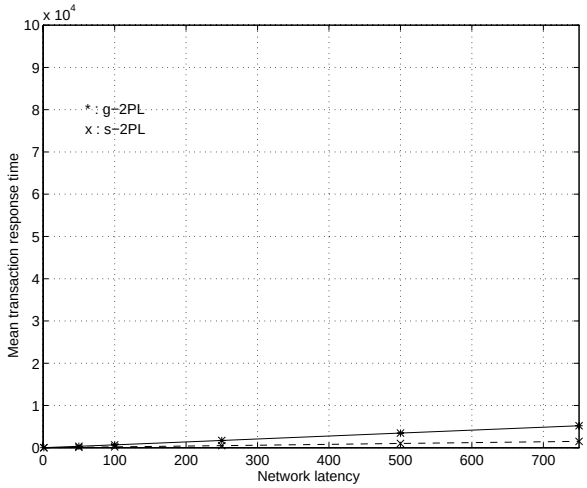

Figure 4: Mean transaction response time of g-2pl & s-2pl versus network latency, p_r=1.0

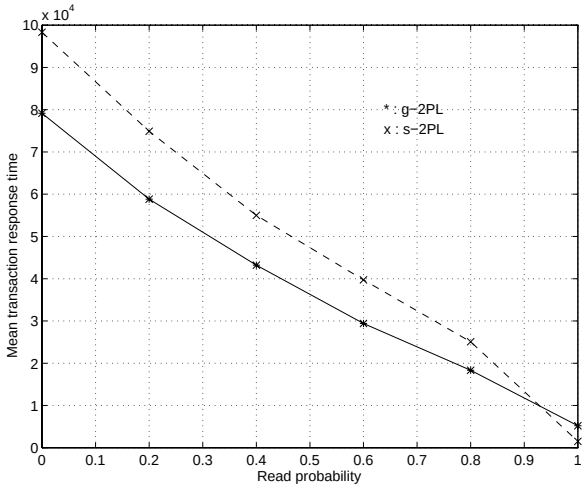

Figure 7: Mean response time of g-2pl & s-2pl versus p_r in l-WAN

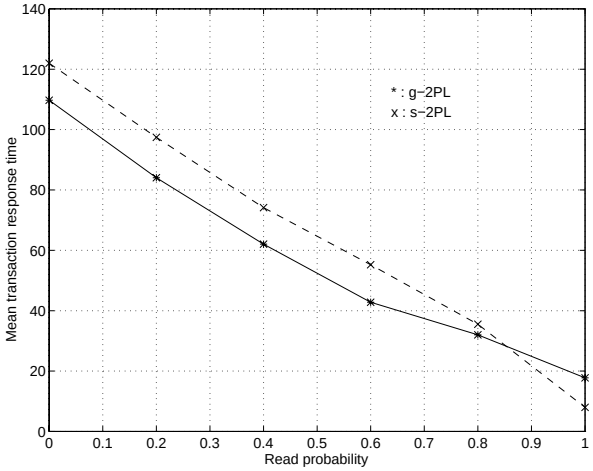

Figure 5: Mean response time of g-2pl & s-2pl versus p_r in ss-LAN

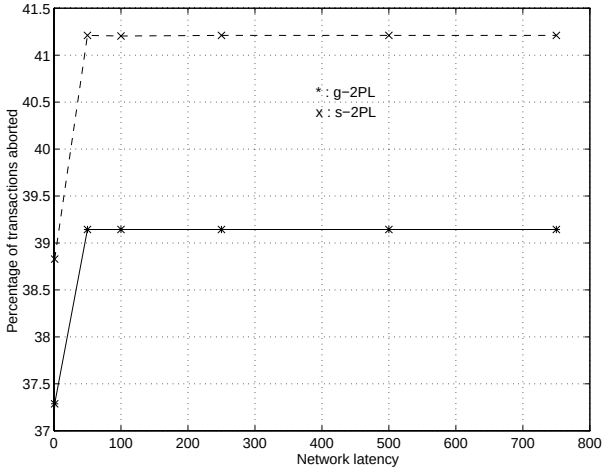

Figure 8: Percentage transactions aborted in g-2pl & s-2pl, p_r=0.6, 50 clients and 25 data items

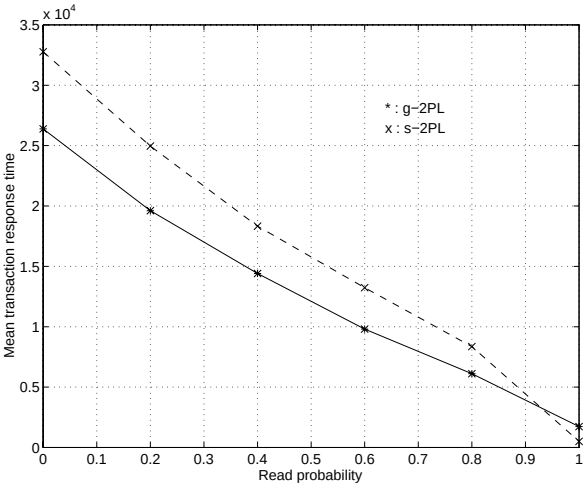

Figure 6: Mean response time of g-2pl & s-2pl versus p_r in MAN

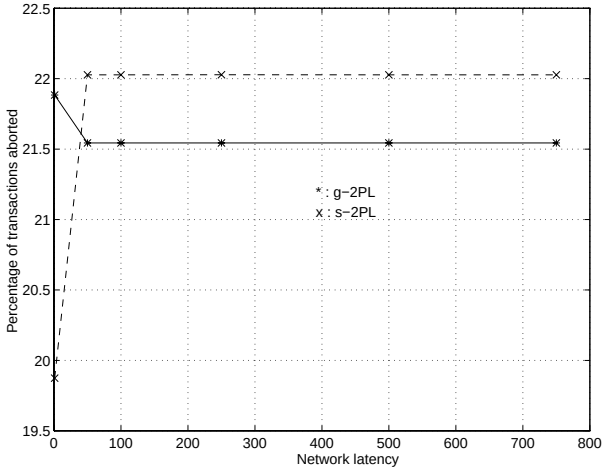

Figure 9: Percentage transactions aborted in g-2pl & s-2pl, p_r=0.8, 50 clients and 25 data items

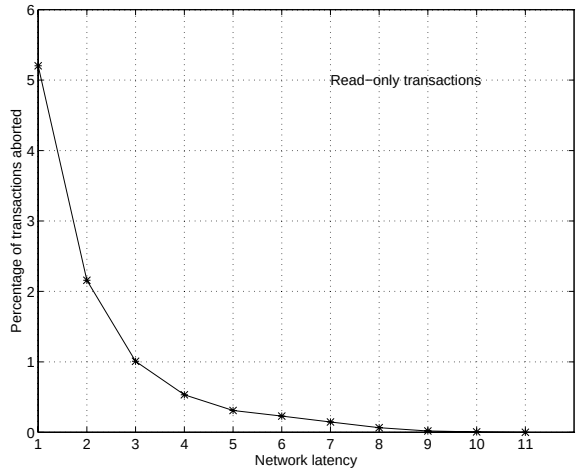

Figure 10: Percentage of transactions aborted as a function of the network latency in a read-only system

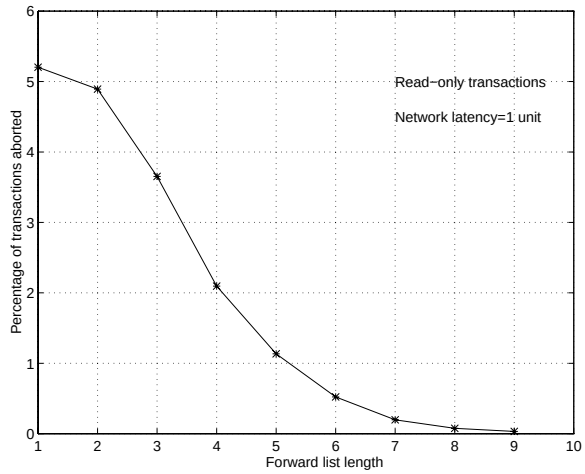

Figure 11: Percentage of transactions aborted versus window size, $p_r = 1.0$ in ss-LAN

scalability to WANs. From Figures 2 – 4, it is evident that only when the read probability is 1.00 (Figure 4) is the performance of s-2PL better than the g-2PL protocol. In the other cases, over the entire range of network latency, g-2PL outperforms s-2PL. The percentage improvement in the response time of the g-2PL protocol over that of the s-2PL protocol was observed to be between 19.50% and 26.92% in the presence of update transactions. The reason for the better performance of s-2PL in read-only systems is that in the g-2PL protocol described here, access requests are granted only at the end of the window periods, and not in between. Thus, the reads are penalized in the g-2PL system and the s-2PL protocol has better performance[2].

To obtain another perspective on the performance comparison, Figures 5 – 7 contain plots of the average transaction response time versus the read probability for 3 different values of the network latency in simulation time units (see Table 2). At low read probabilities, the g-2PL protocol outperforms the s-2PL protocol by grouping access requests and saving on the number of rounds and only at very high read probabilities (close to 1.0) is the performance of the s-2PL protocol better. As the read probability is increased, a cross-over in performance is observed. At the smallest value of the network latency (= 1 unit), the read probability at which the cross-over occurs is high (around 0.85). Further, the cross-over point seems to shift to the right at higher values of the network latency, indicating that in WANs, the performance of the g-2PL protocol is superior to that of the s-2PL protocol over almost the entire range of read probabilities.

[2] In read-only systems, the average transaction response time for transactions accessing a single data item in the s-2PL protocol should be the round-trip network latency.

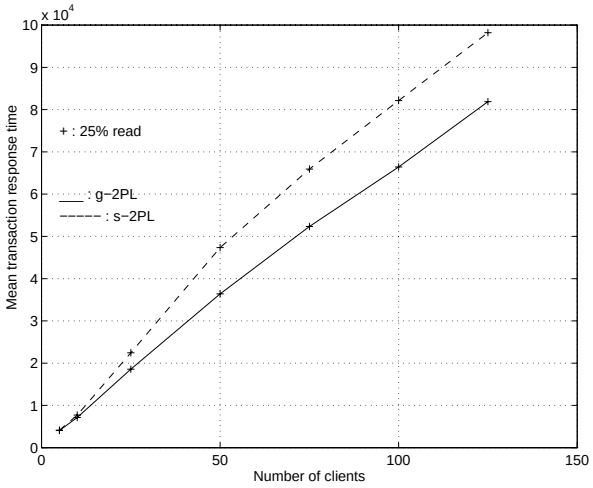

Figure 12: Mean response time versus number of clients: 25 data items, $p_r = 0.25$ in s-WAN

The g-2PL and s-2PL protocols both suffer from the transaction deadlock phenomenon which results in transaction aborts. Thus it is important to compare the percentage of transaction aborts in both protocols, as a function of the network latency and the read probability. Figures 8 – 9 contain the percentage of transactions aborted versus the network latency, for p_r values of 0.6 and 0.8 respectively. The trends are similar for the other values of p_r and hence are not presented here. As expected, the percentage of transactions aborted decreases with increase in the read probability. The percentage of transactions aborted in both protocols is fairly close, although the g-2PL protocol outperforms the s-2PL protocol in the entire range of network latency values studied despite its unique problem of read-only deadlocks (as described in Section 3.3). Further, the percentage of transactions aborted stays fairly constant for all latencies above the single segment LAN case.

When the read probability is high (p_r=0.8) as in Figure 9, for the single segment LAN, the percentage of aborted transactions is unexpectedly high for the g-2PL protocol and the read deadlocks may be the cause. To study this issue further, in Figure 10, the percentage of transactions aborted is plotted versus the network latency in a read-only system. The fraction of transactions aborted due to read-deadlocks decreases with increase in the network latency, and is negligible beyond a network latency of 10 units in

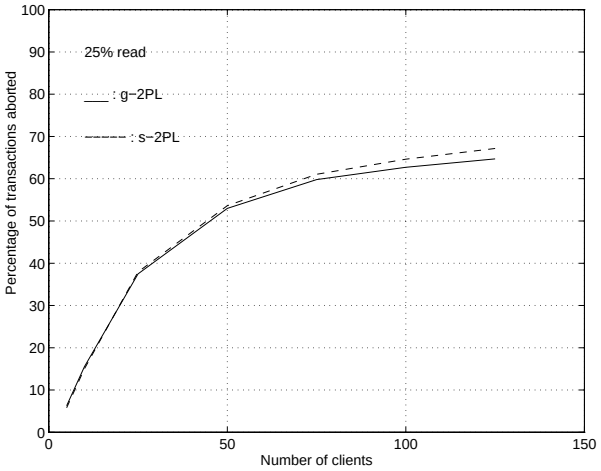

Figure 13: Percentage transactions aborted vs. number of clients: 25 data items, $p_r = 0.25$ in s-WAN

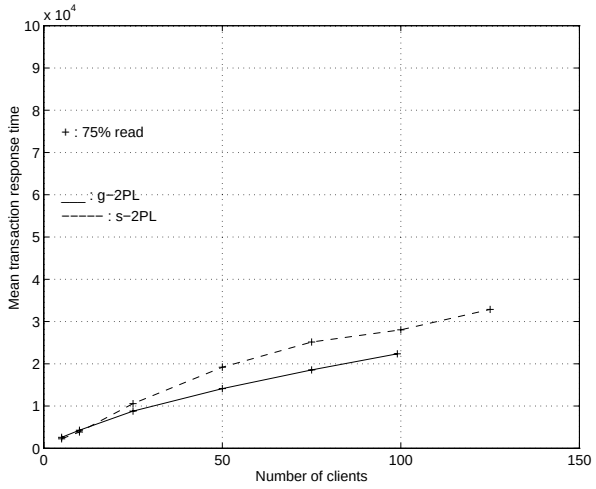

Figure 14: Mean response time versus number of clients: 25 data items, $p_r = 0.75$ in s-WAN

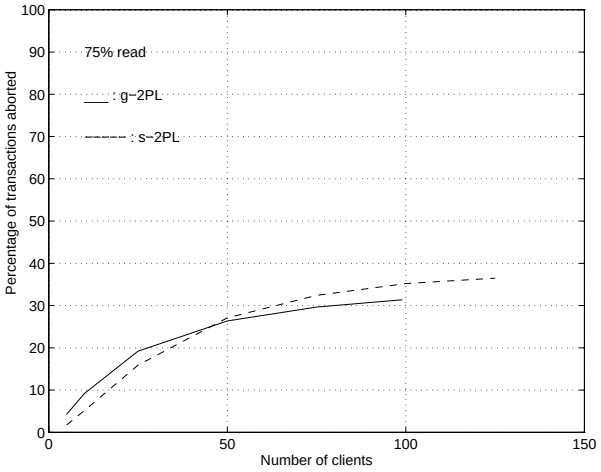

Figure 15: Percentage transactions aborted vs. number of clients: 25 data items, $p_r = 0.75$ in s-WAN

the experiments conducted. The percentage of transactions aborted due to read-deadlocks is never more than a little over 5%. Thus the impact of the read-deadlocks is small and dominant only in the LAN environment. Further, since we have demonstrated that the g-2PL protocol is particularly suited to the WAN environments where the network latency is high, the effect of read-deadlocks can be ignored. The above observation can be explained as follows. With sequential transaction execution patterns (as has been assumed in the system model), at high network latencies, the data requests at the server are spread out over time, causing less conflicts across multiple windows, leading to fewer deadlocks. At a lower network latency, data requests by different transactions occur close together, causing more transaction conflicts in a smaller time frame.

A large collection window allows for the reordering of requests to reduce the deadlock probability. To study this effect we controlled the collection window in terms of the length of the forward list. In Figure 11, the percentage of transactions aborted is plotted against the forward list length in a single segment LAN. The fraction of transactions aborted decreases as the forward list length increases. Beyond a forward list length of 5 requests, the fraction of transactions aborted is less than 1%.

Finally, we studied the effects of increased data contention. The loading on the database system can be increased in several ways. We chose to do so by increasing the number of clients while keeping the transaction profile the same: each transaction uniformly accesses between 1 and 5 data items out of 25 hot data items. The network latency is fixed at 500 time units (small WAN). Figure 12 contains the plots of the average transaction response time for the g-2PL and s-2PL protocols versus the number of clients, with a fixed read probability of 0.25. Figure 14 contains similar plots for a read probability of 0.75. In both cases (p_r=0.25 and p_r=0.75), the g-2PL protocol outperforms the s-2PL protocol at high loads. In the system model described in Section 4, deadlocks are the only cause for transactions to be aborted, i.e., no communication or site failures are assumed. Figures 13 and 15 contain the plots of the fraction of transactions aborted in the g-2PL and s-2PL protocols for read probabilities of 0.25 and 0.75 respectively. From these figures it is evident that the fraction of transactions aborted in both protocols is close. However, it can be seen that at both values of p_r, a cross-over in performance occurs and beyond a certain loading, a higher fraction of transactions are aborted in the s-2PL protocol.

6 Conclusions

Recognizing propagation latencies as the bottleneck and that migrating large amounts of data between clients and servers will not be a problem in future WANs, in this paper we have derived, from the basic server-based two-phase locking protocol (s-2PL), a new protocol called the group two-phase locking protocol (g-2PL) targeted for gigabit-networked client-server systems. In order to study the performance of g-2PL, we have implemented a simulator of a shared nothing, data-server system. In this paper, we reported on the performance of g-2PL in the absence of communication and site failures by comparing it with the performance of s-2PL. The g-2PL scalability has been evaluated with respect to the network latency and the number of clients.

The results of our experiments confirmed our hypothesis that g-2PL is particularly suited to control access to hot data items and showed that g-2PL, in general, outperforms s-2PL for update transactions. Specifically, g-2PL exhibits superior performance when the percentage of reads performed by transactions is relatively low compared to the writes in the database system and the network latency is high. Between 20-25% improvement in the response time was observed. Interestingly, g-2PL exhibits worse response time for read-only transactions although one might have expected that both g-2PL and s-2PL would have behaved the same. This shows that g-2PL unnecessarily penalizes read operations, failing to fully explore the commutativity of read operations as in the case of s-2PL.

As part of our future research, we would like to investigate the performance of g-2PL protocol in the context of read-only transactions by applying the read-only optimization mentioned in this paper. Further, we would like to investigate ways to enhance its performance by considering the various ordering disciplines in forming the forward lists, i.e., the ordering of lock granting within a group. Finally, we would like to extend our simulator in order to compare the g-2PL protocol with more caching protocols.

Acknowledgments: We thank the following students : H. Collier, L. Theophanous and J. Tam, and especially J. Sen and P. Desai, who designed the simulator. This work has been partially supported by NSF awards IRI-9502091 and NCR-9624125 and CRA Distributed Mentor Program awards.

References

[1] K. Wilkinson and M. A. Neimat, "Maintaining Consistency of Client-Cached data," *Proc. of the Intl. Conf. on Very Large Databases*, pp. 122–134, 1990.

[2] Y. Wang and L. Rowe., "Cache Consistency and Concurrency Control in a Client/server DBMS Architecture," *Proc. of the ACM SIGMOD*, pp. 367–376,1991.

[3] M. Carey, M. Franklin, M. Livny, and E. Shekita., "Data Caching Tradeoffs in Client-Server DBMS Architectures," *Proc. of the ACM SIGMOD*, pp. 357–366, 1991.

[4] M. J. Franklin, M. Carey, and M. Livny, "Local Disk Caching for Client-Server Databases," *Proc. of the Intl. Conf. on Very Large Data Bases*, pp. 641–654, 1993.

[5] M. J. Franklin and M. Carey, "Client-Server Caching Revisited," *Distributed Object Management* (T. Ozsu, U. Dayal, and P. Valduriez, eds.), pp. 57–78, 1993.

[6] A. Delis and N. Roussopoulos, "Management of Updates in the Enhanced Client-Server DBMS," *Proc. of the Intl. Conf. on Distributed Computing Systems*, 1994.

[7] A. Billiris and J. Orenstein, "Object Storage Management Architectures," in *Advances in Object-Oriented Database Systems* (A. Dogac, M. T. Ozsu, A. Billiris, and T. Selis, eds.), pp. 185–200, 1994.

[8] P. Bernstein and E. Newcomer, *Principles of Transaction Processing for the Systems Professional*, 1997.

[9] T. Nguyen and V. Shrinivasan, "Accessing Relational Databases from the World Wide Web," *Proc. of the 1996 ACM SIGMOD*, pp. 529–539, 1996.

[10] D. Jadav, M. Gupta, and S. Lalshmi, "Caching Large Database Objects in Wed Servers," *Proc. of the Workshop on Research Issues on Data Engg.*, pp. 10–19, 1997.

[11] M. Hornick and S. Zdonik, "A shared, segmented memory system for an object-oriented database," *ACM Trans. on Office Information System*, vol. 5, no. 1, pp. 70–95, 1987.

[12] C. Lamb, G. Landis, J. Orenstein, and D. Weinreb, "The ObjectStore Database System," *Communication of the ACM*, vol. 34, no. 10, pp. 34–48, 1991.

[13] M. Franklin, M. Zwilling, C. Tan, M. Carey, and D. DeWitt, "Crash Recovery in Client-Server EXODUS," *Proc. of the ACM SIGMOD*, pp. 165–174, 1992.

[14] K. P. Eswaran, J. Gray, R. Lorie, and I. Traiger, "The Notion of Consistency and Predicate Locks in a Database System," *Communications of the ACM*, vol. 19, pp. 624–633, 1976.

[15] J. Howard, M. Kazar, S. Menees, D. Nichols, M. Satyanarayanan, R. Sidebotham, and M. West, "Scale and Performance in a distributed File system," *ACM Trans. on Computer Systems*, vol. 6, no. 1, pp. 51–81, 1988.

[16] M. Carey and M. Livny, "Conflict detection tradeoffs for replicated data," *ACM Trans. on Database Systems*, vol. 16, no. 4, pp. 703–746, 1991.

[17] C. Mohan and I. Narang, "Recovery and Coherency-Control Protocols for Fast Intersystem Page Transfer and Fine-Granularity Locking in a Shared Disks Transaction Environment," *Proc. of the Intl. Conf. on Very Large Data Bases*, 1991.

[18] S. Banerjee and P. K. Chrysanthis, "A Fast and Robust Failure Recover Scheme for Shared-Nothing Gigabit-Networked Databases," *Proc. of the Intl. Conf. on Parallel and Distributed Computing Systems*, pp. 684–689, 1996.

[19] L. Kleinrock, "The Latency/Bandwidth Tradeoff in Gigabit Networks," *IEEE Communications Magazine*, vol. 30, pp. 36–40, 1992.

[20] S. Banerjee and P. K. Chrysanthis, "Data Sharing and Recovery in Gigabit-Networked Databases," *Proc. of the Intl. Conf. on Computer Communications and Networks*, pp. 204–211, 1995.

[21] P. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*, 1987.

[22] J. N. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*, 1993.

[23] D. DeWitt, R. Katz, F. Olken, L. Shapiro, M. Stonebraker, and D. Wood, "Implementation Techniques For Main Memory Database Systems," *Proc. of the ACM SIGMOD*, pp. 1–8, 1984.

[24] D. Gawlick and D. Kinkade, "Varieties of Concurrency Control in IMS/VS Fast Path," *IEEE Database Engineering*, vol.8, 1985.

[25] S. Banerjee and P. K. Chrysanthis, "A New Token Passing Distributed Mutual Exclusion Algorithm," *Proc. of the Intl. Conf. on Distributed Computing Systems*, pp. 717–724, 1996.

[26] S. Banerjee and P. K. Chrysanthis, "Performance evaluation of the g-2PL protocol," *Proc. of the Intl. Conf. on Parallel and Distributed Computing Systems*, pp. 428–432, 1997.

[27] S. Banerjee and P. K. Chrysanthis, "Network Latency Optimizations in Distributed Database Systems," TR-97-27, Computer Science, University of Pittsburgh, 1997.

[28] R. Jain, *The Art of Computer Systems Performance Analysis*. 1991.