PAPER *Special Issue on New Generation Database Technologies*

Group Two-Phase Locking: A Scalable Data Sharing Protocol

Sujata Banerjee[†] *and* **Panos K. Chrysanthis**[††], *Nonmembers*

SUMMARY The advent of high-speed networks with quality of service guarantees, will enable the deployment of data-server distributed systems over wide-area networks. Most implementations of data-server systems have been over local area networks. Thus it is important, in this context, to study the performance of existing distributed data management 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. In this paper, we examine and compare the scalability of the server-based two-phase locking protocol (s-2PL), and the group two-phase locking protocol (g-2PL). The s-2PL protocol is the most widely used concurrency control protocol, while the g-2PL protocol is an optimized version of the s-2PL protocol, tailored for high-speed wide-area network environments. The g-2PL protocol reduces the effect of the network latency by message grouping, client-end caching and data migration. Detailed simulation results indicate that g-2PL indeed scales better than s-2PL. For example, upto 28% improvement in response time is reported.

key words: Distributed Databases, Gigabit Networks, Concurrency Control, Data Caching, Transaction Processing, Client-Server Configuration

1. Introduction

The rapid growth of Internet applications and the World Wide Web (WWW), has spurred the need for high performance distributed information/data-server systems over wide area networks (WANs). To achieve the goals of such a high performance system, the network speed is being significantly increased, particularly in the WAN environment. Further, 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. 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 WANs. 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. 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]. However, a key problem that is yet to be addressed adequately is the development of scalable data-server protocols that can take advantage of the enhanced infrastructure, which is the topic of this paper.

Up until recently, the network speed in local area networks (LANs) was considerably higher than in WANs, but that is no longer true, making the data transmission delays smaller in both LANs and WANs. Clearly, minimizing message sizes in order to reduce transmission delays is no longer that important. However, a significant difference that still exists is the relatively high network latency in WANs due to larger physical distances and hence larger signal propagation latencies and the possibility of queuing delays at each intermediate switching hop. In a high-speed LAN environment, the network latency is of the order of microseconds while it is of the order of milliseconds in a high-speed WAN environment. In fact, the network latency in a high speed WAN is the dominant component of the transmission [11], processing or I/O overheads. Thus the initial data server systems were deployed over 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], [12]–[15]. These LAN data-server systems have exhibited promising performance levels but for a limited number of clients [6], [16], [17]. When data-servers, as well as collaborating-servers, become available over high speed 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. In order to fulfill these expectations and obtain *scalable* performance, we need to combat the high network latency. Further, unlike other overheads the only possible way for reducing the effects of high latencies on the performance of database systems, is to hide them in *innovative* data management protocols that supports data migration. At the same time, it is not practical to overhaul existing algorithms that are widely in use, and our choice is the path of evolution starting with the modification and/or optimization of existing schemes with the new system assumptions.

Client caching of data items and locks is a popular mechanism that has been proposed to reduce the data transmission overheads. In LAN environments, 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) [18], namely, *Server-based 2PL, Optimistic 2PL* and *Callback Locking* [1], [2], [13], [14], [19], [20]. While caching can significantly improve performance [17], the marginal gains decrease rapidly as the network speed is increased and when data items are frequently updated rendering the caches invalid. In fact, the server-based 2PL (s-2PL) protocol was found to have the best performance in situations with high data contention in LANs [5]. Further, in a

[†]Information Science & Telecommunications dept.
[††]Computer Science dept., Univ. of Pittsburgh.

high-speed WAN environment, due to the above-mentioned shifts in the overhead, efforts should be focused on reducing the number of sequential message passing rounds (since each round can incur a significant delay, even if the transmission time is negligible) rather than the data transmission time. In this paper, a strategy to reduce message passing rounds is described that scales better than the s-2PL protocol under high data contentions in a high-speed WAN. Our strategy, embodied in the *group 2PL* (g-2PL) protocol assumes a stronger inter-client cooperation, intra-transaction caching at the clients and message grouping.

In this paper, we present an enhanced version of the original g-2PL protocol [21], [22], and emphasize its scalability properties. In the next section, we propose various protocol scalability metrics with respect to the geographical span of the network, the number of clients and the data contention, in terms of which we perform our evaluation. Then the enhanced g-2PL protocol and the s-2PL protocol are described in Section 3. Using simulation, the scalability metrics for the g-2PL and s-2PL protocols are compared. Section 4 consists of a description of the simulation testbed, which is followed by a presentation of the numerical results in Section 5. 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 improvement in response time is significant at about 28%. Section 6 concludes the paper with a discussion of future research.

2. Scalability Metrics

A key issue in the development of next generation data-server systems is protocol performance scalability. For example, as the system size increases with a larger number of clients accessing the database, the increased data contention degrades the performance. Thus it is imperative to compare the two protocols with respect to the number of clients each can support, given a specific performance criterion, such as the average transaction response time or transaction throughput. Further, scalability issues with respect to the network span and the data contention levels must also be studied. We propose the following three scalability metrics.

2.1 Scalability with respect to Network Span

As the geographical span of a network increases, the network latency increases correspondingly. Depending on the capability of the data server protocol to hide this latency, the protocol may or may not scale with the network geographical span. For instance, a specific protocol may not scale to a WAN environment if there is a requirement to meet an upper bound on the average transaction response time.

Geographical scalability (GS) in the context of this paper is defined as the percentage larger network span supported by g-2PL as compared to s-2PL, for a specific upper bound on the average transaction response time.

$$\text{GS}(\tau) = \frac{d_g(\tau) - d_s(\tau)}{d_s(\tau)}, \qquad (1)$$

where, d_g and d_s are the network latencies at which the g-2PL and s-2PL protocols can provide the same average transaction response time τ.

2.1.1 Scalability with respect to data contention

Given a specific network geographical span, increasing the data contention by decreasing the read probability causes the average transaction response time. Thus it is important to study the levels of data contention that can be supported by a specific protocol, again given a bound on the average transaction response time. The data contention scalability (DCS) is defined as the ratio of the contention levels that g-2PL can support as compared to the the s-2PL protocol, for a given upper bound on the average transaction response time. The data contention level is inversely proportional to the read probability.

$$\text{DCS}(\tau) = \frac{p_r^s(\tau)}{p_r^g(\tau)}, \qquad (2)$$

where, p_r^s and p_r^g are the read probabilities at which the s-2PL and g-2PL protocols can provide the same average transaction response time τ.

2.1.2 Scalability with respect to the number of clients

Finally, given a read probability and a network geographical span, the average transaction response time increases with the number of clients, which increases the system size. The system scalability (SS) is defined as the percentage increase in the number of clients that can be supported by g-2PL compared to the s-2PL, given a specific bound on the average transaction response time.

$$\text{SS}(\tau) = \frac{c_g(\tau) - c_s(\tau)}{c_s(\tau)}, \qquad (3)$$

where, c_g and c_s are the number of clients that can be supported by the g-2PL and s-2PL protocols while providing the same average transaction response time τ.

3. The Data Access Protocols

In this section, first the s-2PL protocol is briefly reviewed. Then, an optimized version of the s-2PL protocol, the group two-phase locking (g-2PL) protocol is described, and quantitatively compared to the s-2PL protocol in the following section. The choice of comparing/optimizing the g2Pl protocol with the s-2PL protocol rather than any other concurrency control protocol is motivated by two reasons. The first is that the s-2PL protocol has been shown to have the best performance in situations with high data contention in LANs [5]. The second reason is the wide-spread deployment of the s-2PL protocol in existing database systems because of its relative simplicity in implementation. Thus, by optimizing s-2PL, the maximum impact on performance as well as widespread acceptance can be expected. Although we do not deal with the recovery aspects of the g-2PL protocol here, it is assumed that the sites follow the standard protocol adopted by the s-2PL protocol where each site uses *Write-Ahead Logging* (WAL) and garbage collects its log once the data are made permanent at the server [23].

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 [18]. 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 [24], [25].

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], [14], [17]. To simplify the discussion, in the rest of the paper we focus only on the s-2PL protocol but the results can be 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.

If a transaction needs to access n data items, the first phase of the protocol as 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 Group Two-Phase Locking Protocol

The core of the g-2PL protocol [22] is to apply *grouping of messages sent to multiple sites* to the s-2PL protocol, thus reducing the message passing rounds. Grouping of actions involving a single site has been previously used successfully in other situations (e.g., in group commit [26], [27] multiple transactions are committed and acknowledged at a single site). Specifically, the lock (data) granting and release messages are grouped 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. 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.

Thus 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.

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 collects requests as the *collection window* for the data item[†]. Once the lock is returned and a data-server receives and installs the new version of a data item in the database, the previous collection period ends, a new forward list is created, using which, the server dispatches the data item to the first client on the new forward list. Initially at start-up time and during periods of extremely light loading, the forward-list will contain a single client.

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. 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

[†]Our experimentation with a tunable collection window size and a timeout proved that tuning the collection window does not produce significant performance gains [21].

1.	First-in-First-Out or sort by arrival of the requests.
2.	Order by the client ID.
3.	Order by transaction priority.
4.	Order by the number of locks held by each transaction. • a. Transactions with fewer number of locks go first. • b. Transactions with greater number of locks go first.
5.	Serve the read requests first.
6.	Split up the read requests according to the multi-programming capabilities.
7.	Order requests such that the total distance traversed by the messages is minimized.

Table 1 Ordering rules for the forward list

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. This is termed the MR1W (*Multiple Reads One Write*) optimization. With this optimization the g-2PL protocol behaves similar to the two-copy version s-2PL protocol [24] which allows more concurrency than the standard s-2PL protocol.

To improve performance further, the forward list for each data item may be created according to one of several ordering rules (See Table 1). The default rule is FIFO or sort by arrival of the request as in the s-2PL protocol. The effect of ordering rules are evaluated in Section 5. However, we next describe a deadlock-avoidance FL ordering rule that is an integral part of the g-2PL protocol.

3.2.1 Deadlock avoidance by FL reordering

Two-phase locking protocols are susceptible to deadlocks [18], 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. This typically occurs for read-write and write-write conflicts. In addition to the above deadlocks, the g-2PL protocol is susceptible to a unique type of deadlock which is created due to *read-only* dependencies formed across different collection windows. This is described further in Section 3.2.2. 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 [24], [25]. However, in the case of the g-2PL protocol, some deadlocks can be avoided by intelligently creating the forward lists.

Specifically, some 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. The precedence graph is consistent with the lock granting order and hence 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 increases the utilization of data-server CPU while reducing the transaction response time.

3.2.2 Issues with Read-only transactions

There are two related issues associated with read-only transactions and the g-2PL protocol. The first concerns a potential deadlock situation caused by *read-only* dependencies and the other is the response time of read-only transactions. This potential deadlock situation is better illustrated using an example. In Section 5, we shall demonstrate that these read-only deadlocks occur rarely and only under some special circumstances.

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 the opposite order. As soon as the data-server gets the requests $read_1(x)$ and $read_2(y)$, it 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. The only way to resolve the situation is to abort one of the transactions.

The second issue concerns the response time of read-only accesses. The data server responds to the next set of requests only when the data item (and lock) is returned back to it by the previous set of clients. This can unnecessarily delay read requests that have no conflict with the previous requests. A solution to both of these problems is as follows.

A forward list that contains only read requests is termed a read-only forward list (RO-FL). Any read request received by the server after it has dispatched a RO-FL, is granted immediately without waiting for the data item to return, thus removing any extra delays. This is equivalent to expanding

the RO-FL to include new read requests that arrive after the forward list is constructed and transmitted. This solution will also remove all *read-only* dependencies in a read-only system. Consider the above example where transactions t_1 and t_2 request read access to x and y in the opposite order. Knowing that the forward lists for x and y are read-only, the server will grant the new requests as if they have arrived in the previous window. The initial forward lists for data items x and y are as below.
Initial FL for x: $read_1(x)$ and Initial FL for y: $read_2(y)$.
Expanded FL for x: $read_1(x)$,$read_2(x)$ and Expanded FL for y: $read_2(y)$,$read_1(y)$.

This modification is easy to implement and can be handled by the server without contacting the clients of t_1 or t_2. The forward list can be expanded as long as consecutive read requests arrive. A new forward list will be started with the arrival of a write request. The server will wait for acknowledgments from all reading clients that are on the expanded forward list before dispatching the next forward list.

4. System Model for Performance Evaluation

In order to evaluate the performance of the s-2PL and g-2PL protocols under a high-speed networking environment, 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, and. no site and communication failures occur.

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. Two scenarios are considered when a transaction is aborted. In the first scenario, each transaction that is aborted is replaced by another independent transaction. In the second scenario which is more realistic, each transaction that is aborted is restarted by another transaction and the overall transaction response time includes the time overhead of restarting. Both scenarios are evaluated to study the impact of grouping on restarts. The simulation model assumes that the computation cost at the data server to reorder the forward lists as well as computing the wait-for-graphs is the same.

Table 2 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 wide area high speed networking 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 0.5 and 375 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 local area networks (LAN) to wide area networks (WAN), as listed in Table 3 with their 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 cluster of Sun Ultra processors with the Solaris 2.5.1 operating system. The transient phase of the simulation runs was eliminated. In each simulation run, 100,000 transactions (excluding the transient phase) were generated, requiring a simulation time of upto 100 million time units. 95% confidence intervals on the average transaction response time were calculated from 5 independent simulation runs. The relative precision of the transaction-level measurements never exceeded 2% of the mean values. To conserve space, only the salient results are presented.

Before describing the main results of this study, first

# Servers	1
% read accesses	0 – 100 %
# Clients	varying
Network Latency	1 – 750 time units
# hot data items	25
Computation Time per operation	1 – 3 time units
Transaction Execution Pattern	Sequential
Idle Time between transactions	2 – 10 time units
# data items reqd. by a transaction	1 – 5
Multiprogramming level at clients	1

Table 2 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 3 Networking Environments Simulated

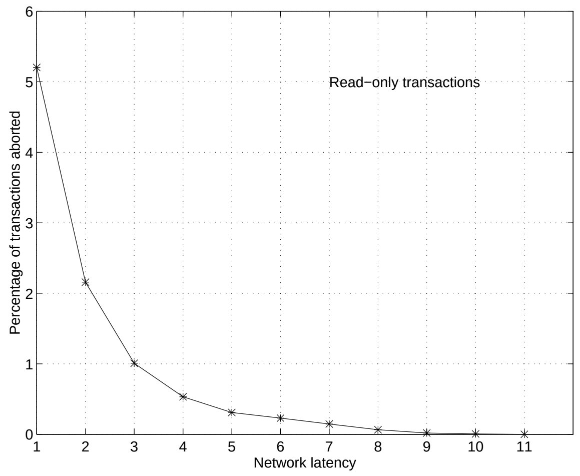

Fig. 1 Percentage of transactions aborted as a function of the network latency in a read-only system

the impact of read-dependencies is assessed. In Figure 1, 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 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. 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. Given that the impact of the read deadlocks is found to be minimal, and the focus of this study is on high data contention environments, the solution proposed in Section 3.2.2 to remove these deadlocks was not implemented in the simulation model, bearing in mind that without this solution, read accesses across multiple windows may suffer a larger delay.

The g-2PL protocol is particularly suited to accessing hot data items. Thus we simulated the hot portion of a database 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. For each protocol, there are two curves, one that simulated the restart process (the higher values), and another that did not. Obviously as the network latency is increased, the average transaction response time increases correspondingly. 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. Without taking into account the restart process, 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 percentage improvement in the response time including the restart overhead of the g-2PL protocol over that of the s-2PL protocol is observed to be between 18% and 28% in the presence of update transactions. Thus, the maximum improvement of the g-2PL protocol over the s-2PL protocol increases by approximately 1-2% when restarts are taken into account. 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[†].

The above results are with the FIFO ordering of the forward list. The effect of three re-ordering schemes (ordering rules 1, 4b and 5 from Table 1) on the performance are presented in Table 4 for the above system parameters. As seen from these results, the effect of the various re-ordering schemes are minimal (less than 1% from the FIFO case). Thus the remainder of the results are presented for the FIFO ordering of the FL, and including the effect of restarts.

From the above graphs, the geographical scalability (GS(τ)) metric can be computed. This is shown in Table 5 in which the geographical scalability is tabulated for a given bound on the average transaction response time. This scalability measure is independent of the actual response time bound used. This table shows that the g-2PL protocol is 25% more scalable than the s-2PL protocol when $p_r = 0.0$ and 36.36% when $p_r = 0.6$. On the other hand, in a read-only system ($p_r = 1.0$), the s-2PL protocol is approximately

[†]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.

Campus Area Network

p_r	FIFO (Rule 1)	Rule 4b	Rule 5
0.0	10582.968750	10558.753906	10608.459961
0.4	5678.985840	5672.341797	5652.992676
1.0	691.264709	693.854187	693.949646

Large Wide Area Network

p_r	FIFO (Rule 1)	Rule 4b	Rule 5
0.0	79249.976562	79315.117188	79301.273438
0.4	42604.316406	42596.437500	42529.902344
1.0	5209.281738	5176.771973	5200.751465

Table 4 Mean transaction response time of g-2pl for three ordering rules

p_r	GS(τ)
0.00	25.00 %
0.60	36.36 %

Table 5 Geographical network scalability versus p_r

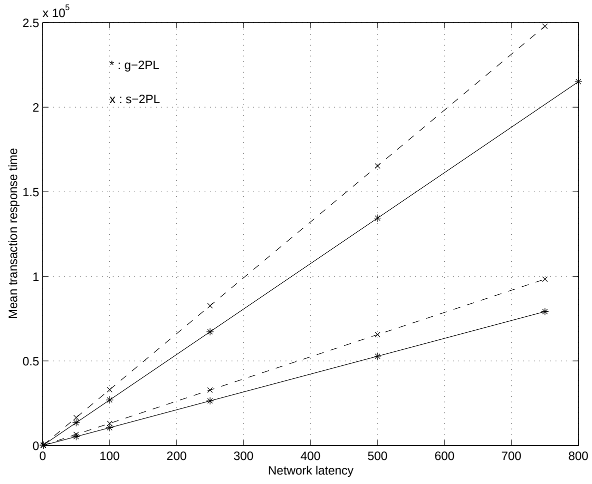

Fig. 2 Mean transaction response time of g-2pl & s-2pl versus network latency, p_r=0.0

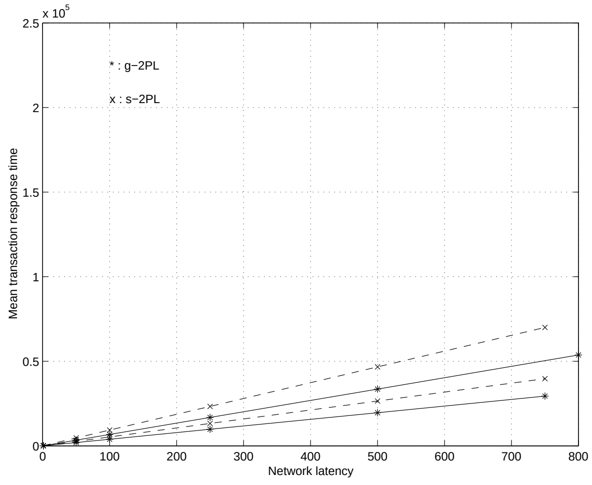

Fig. 3 Mean transaction response time of g-2pl & s-2pl versus network latency, p_r=0.6

twice as scalable with network span as the g-2PL protocol.

Figure 5 contain plots of the average transaction response time versus the read probability for a network latency of 250 simulation time units. 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.

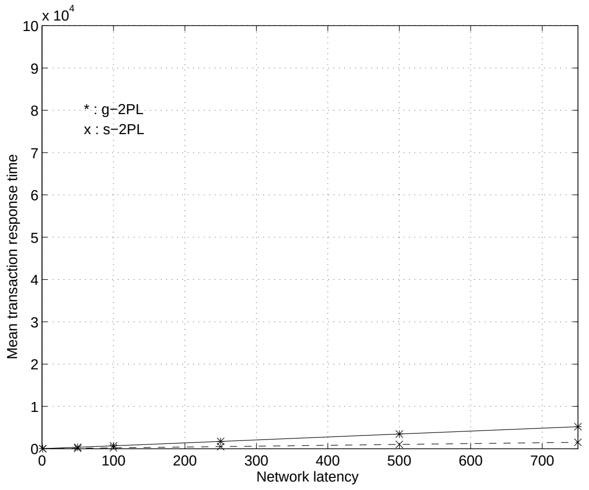

Fig. 4 Mean transaction response time of g-2pl & s-2pl versus network latency, p_r=1.0

As can be seen from the figures, the average transaction response time decreases non-linearly as the data contention levels are decreased. Thus the data contention scalability (DCS(τ)) depends on the actual average transaction response time bounds considered. The g-2PL protocol is found to be upto 3 times more scalable than the s-2PL protocol with respect to data contention. This implies that the g-2PL protocol can support upto three times the data contention levels that the s-2PL protocol can support.

To compute the system scalability (SS(τ)) metric, the following experiments were conducted. The network latency is fixed at 500 time units (small WAN) and each transaction accesses between 1 and 5 data items, out of a total of 25 hot data items. Figure 6 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. Except in very small systems, the response time in the s-2PL protocol is higher (upto 28%). Also, in this case as well, the average transaction response time increases non-linearly with the number of clients. Thus, the system scalability is a function of the average transaction response time bound chosen. The g-2PL protocol is found to be capable of supporting upto 26% more clients than the s-2PL protocol when $p_r = 0.25$. At $p_r = 0.75$, the g-2PL can support upto 78.5% more clients than the s-2PL protocol for the parameter ranges studied. Again, note that the scalability can be expected to be larger than 78.5% in some cases as the deadlock detection scheme in s-2PL is made more realistic.

Figure 7 contains a plot of the average transaction throughput versus the number of clients, for a read probability of 0.75. The s-2PL and the g-2PL protocols thrash at a system size of 8 clients and 62 clients respectively.

The g-2PL and s-2PL protocols both suffer from the transaction deadlock phenomenon which results in transac-

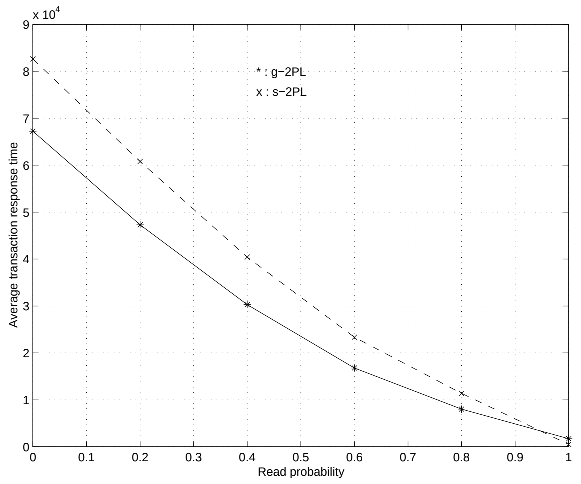

Fig. 5 Mean transaction response time of g-2pl & s-2pl versus the read probability for latency = 250 units (MAN)

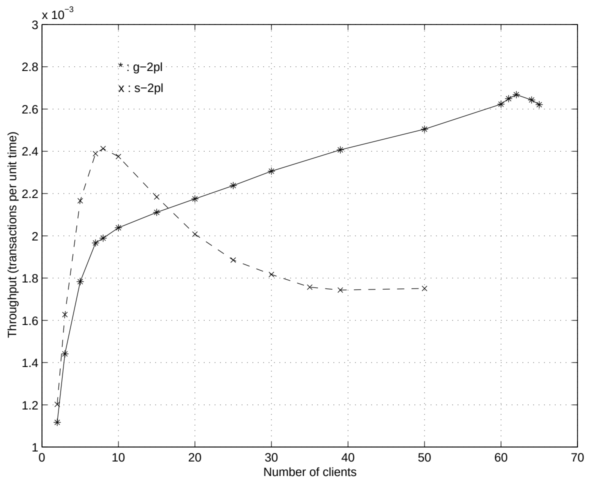

Fig. 7 Mean transaction throughput versus number of clients: $p_r = 0.75$ and network latency = 500 time units

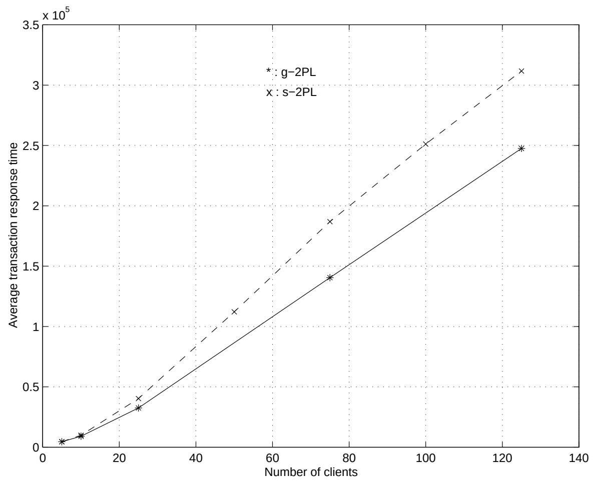

Fig. 6 Mean transaction response time versus number of clients: $p_r = 0.25$ and network latency = 500 time units

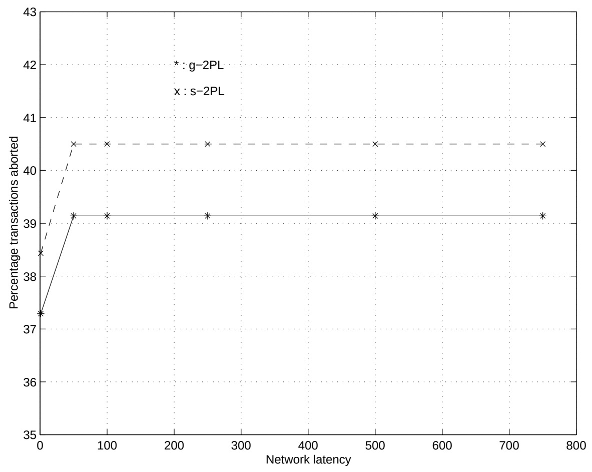

Fig. 8 Percentage of transactions aborted versus network latency: $p_r = 0.6$

tion 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. Due to space restrictions, graphical results are not presented on this topic. 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. Further, the percentage of transactions aborted stays fairly constant (see Figure 8) for all latencies above the single segment LAN case (latency of 1 unit).

6. Conclusions

Traditionally, the performance evaluation of caching schemes in data server systems, which were conducted over local area networks, have not taken scalability issues into account. However, this is a key issue in future systems when data server systems are migrated to wide area networks, with significant increases in data contention, the number of clients and larger network latencies due to the large geographical network span. 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 (s-2PL) protocol, a new protocol called the group two-phase locking (g-2PL) protocol targeted for gigabit-networked client-server systems. In order to study the performance of the g-2PL protocol, we have implemented a simulator of a shared nothing, data-server distributed database system. In this paper, we reported on the performance of the g-2PL protocol in the absence of communication and site failures by comparing it with the performance of the s-2PL protocol.

The results of our experiments confirmed our hypothesis that the g-2PL protocol is particularly suited to control access to hot data items and showed that the g-2PL protocol, in general, outperforms the s-2PL protocol for update transactions. Specifically, the g-2PL protocol 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 18-28% improvement in the response time was observed. Further, the g-2PL protocol is more scalable than the s-2PL protocol. It scales to between 25–36% larger networks, 26–78.5% more clients and upto 3 times the data contention levels than the s-2PL protocol. In addition, although not shown in this paper, it balances the network traffic between the server and the clients so as to reduce the traffic at the server by a factor of between 10–15.

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 discussed in this paper. We would like to extend our simulator along a number of directions, such as to include the fine level CPU processing granularity. This will allow us to compare the g-2PL protocol with more caching protocols.

Acknowledgments

This work has been partially supported by NSF awards IRI-9502091, NCR-9624125, NCR-9702389 and IIS-9812532.

References

[1] K. Wilkinson and M. A. Neimat, “Maintaining Consistency of Client-Cached data,” in *Proc. of the 16th Intl. Conf. on Very Large Databases*, pp. 122–134, Aug. 1990.

[2] Y. Wang and L. Rowe., “Cache Consistency and Concurrency Control in a Client/server DBMS Architecture,” in *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pp. 367–376, May 1991.

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

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

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

[6] A. Delis and N. Roussopoulos, “Management of Updates in the Enhanced Client-Server DBMS,” in *Proc. of the 14th Intl. Conf. on Distributed Computing Systems*, Jun. 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, Springer Verlag, 1994.

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

[9] T. Nguyen and V. Shrinivasan, “Accessing Relational Databases from the World Wide Web,” in *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pp. 529–539, May 1996.

[10] D. Jadav, M. Gupta, and S. Lalshmi, “Caching Large Database Objects in Wed Servers,” in *Proc. of the 7th Intl. Workshop on Research Issues on Data Engineering*, pp. 10–19, Apr. 1997.

[11] L. Kleinrock, “The Latency/Bandwidth Tradeoff in Gigabit Networks,” *IEEE Communications Mag.*, vol. 30, pp. 36–40, Apr. 1992.

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

[13] 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.

[14] M. Franklin, M. Zwilling, C. Tan, M. Carey, and D. DeWitt, “Crash Recovery in Client-Server EXODUS,” in *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pp. 165–174, May 1992.

[15] S. Venkataraman, J. F. Naughton, and M. Livny, “Remote Load-Sensitive Caching for Multi-Server Database Systems,” in *Proc. of the Intl. Conf. on Data Engineering (ICDE)*, Feb. 1998.

[16] A. Delis and N. Roussopoulos, “Performance and Scalability of Client–Server Database Architectures,” in *Proc. of the 19th Int'l Conf. on Very Large Databases*, 1992.

[17] M. J. Franklin, M. Carey, and M. Linvy, “Transactional client-server cache consistency: Alternatives and performance,” *ACM Transactoions on Database Systems*, 1997.

[18] 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, Nov. 1976.

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

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

[21] S. Banerjee and P. K. Chrysanthis, “Performance evaluation of the g-2PL protocol,” in *Proc. of the Tenth Intl. Conf. on Parallel and Distributed Computing Systems (PDCS)*, pp. 428–432, Oct. 1997.

[22] S. Banerjee and P. K. Chrysanthis, “Network Latency Optimizations in Distributed Database Systems,” in *Proc. of the Intl. Conf. on Data Engineering (ICDE)*, pp. 532–540, Feb. 1998.

[23] 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,” in *Proc. of the 17th Intl. Conf. on Very Large Databases*, 1991.

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

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

[26] D. DeWitt, R. Katz, F. Olken, L. Shapiro, M. Stonebraker, and D. Wood, “Implementation Techniques For Main Memory Database Systems,” in *Proc. of the ACM SIGMOD Intl. Conf. On Management of Data*, pp. 1–8, 1984.

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

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