In Proceedings of the 10th Intl. Conf. on Parallel and Distributed Computing Systems (PDCS), pp. 428–432, 1997.

Performance Evaluation of the Group Two-Phase Locking Protocol

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

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

Abstract

In this paper, we discuss the initial results of a detailed simulation of the *group two-phase locking* (g-2PL) protocol which was developed for gigabit-networked databases. These results show that the grouping of lock grants, client-end caching and data migration proposed in the g-2PL protocol improves the transaction response time upto 25% at moderate to high network latencies, when compared to traditional schemes.

1 Introduction

Several exciting advances are being made in the general area of high speed distributed computing. For instance, the rate at which information can be transmitted [11] and the rate at which information can be processed is increasing. 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. As the data rate in wide area networks 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. At gigabit rates in a wide area network, the propagation latency is the dominant component of the overall delay [9]. The minimum coast-to-coast (U.S.) round trip network latency on the internet has been observed to be 70 msec [10].

The above basic characteristic of high speed wide area networks (referred to as a high bandwidth-delay product) has significant implications on distributed applications. Moreover, since bits cannot travel faster than the speed of light, and the distance between communicating computers cannot be reduced, the only way to combat propagation latency is to hide it in innovative protocols. This is not to say that the performance of a traditional distributed 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. Beyond a certain data rate, there will be no further improvement, no matter what the increase in the data rate is, and unless newer database protocols are developed that are *distance-independent*, scalable performance will not be achieved. This observation motivated us to develop a concurrency control protocol [1], and a corresponding recovery algorithm in [2], which are the first ones in the family of algorithms which we refer to as *APLODDS* for Algorithms for Propagation Latency Optimization in Distributed Database Systems.

Our proposed concurrency control protocol in [1], called *group two-phase locking* (or g-2PL), is a variant of the *strict two-phase locking* (s-2PL) protocol [6] and is targeted for client-server distributed database system [5, 12] in a shared-nothing environment. The g-2PL protocol is specially tailored for a gigabit wide area environment in which the size of the message is less of a concern than the number of sequential phases or rounds of message passing. The g-2PL protocol enhances performance of the database system by exploiting these characteristics of gigabit networks and in particular reduces the number of rounds of message passing by grouping the lock grants, client-end caching and data migration. In this paper, the performance of the g-2PL protocol is evaluated using simulation, and compared with the s-2PL protocol, the most widely used concurrency control 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 in general outperforms the s-2PL protocol when the percentage of reads performed by transactions is relatively low compared to the writes in the database system.

The rest of the paper is structured as follows. In Section 2, the g-2PL protocol is described in some detail. The simulation system model is presented in 3, followed by the numerical results of the performance evaluation obtained by simulation in section 4.

2 The g-2PL Protocol

To simplify the presentation of the g-2PL protocol, we consider here a distributed database with a single traditional database (DB) server and multiple clients with local processing capabilities. When a client needs a data item, it sends a request to the DB server which responds with the requested data item. That is, we assume a *data shipping* client-server environment [7].

One of the motivations in a high speed environment is to minimize both the number of messages as well as the rounds. The g-2PL protocol proposes to reduce the number of phases of message passing by *grouping* the lock (data) granting and release. The DB server collects the lock requests for each data item for a specified interval. At the end of this interval (referred to as the *collection window* or simply, the window from now on), the lock is granted to the first transaction, and the data item is sent to the respective client along with the ordered list (also referred to as the *forward list*) of the clients that have pending lock requests for that data

item, that arrived within the window. Within each window, the forward list may be created according to one of several rules (see [1] for details) to improve performance further. While the data items have been sent out to a group of clients, the server continues to collect requests.

When a transaction commits, the client sends the new version of the 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 server with the outcome of each transaction executed on the clients on the forward list. That is, in the 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 compared to the standard s-2PL protocol.

In the description of the basic protocol, only exclusive access to data was considered. Obviously, access to some data may be done in a shared fashion, with multiple clients *reading* the data item simultaneously. Shared access is incorporated into the basic protocol by allowing multiple readers and a single writer to execute concurrently while still preserving strict consistency. For each data item required in the shared mode by multiple (reading) clients, the DB server can send a copy of the data item to each of the reading clients, with the forward list containing the client C_i that requires the data item next in the exclusive mode. At the same time, a message containing the data item and the list of the shared-mode clients is also sent to C_i that requires exclusive access. Although this enables C_i to execute concurrently with the reading clients, C_i cannot release its updates until it receives a *release* message from all the reading clients. Here it is interesting to point out that the protocol just described behaves similar to the two-copy version s-2PL protocol [4] which allows more concurrency than the standard s-2PL protocol. If there are no waiting transactions that need exclusive access, the release messages are returned to the server.

Two-phase locking protocols are deadlock-prone [6], 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. In the case of the g-2PL protocol, deadlocks can be prevented 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, which 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 transaction precedence graph is a directed graph which determines the order in which each data item will *move* from one client site to another. Also, the precedence graph is consistent with the lock granting order and hence consistent with the serialization order. In the case that such reordering of forward lists is not possible, some transactions may have to be aborted.

The rest of the paper discusses a simulation study of the performance of the g-2PL protocol. To keep the paper more focused, several enhancements to the basic protocol [3] have not been discussed. The most important performance aspect in concurrency control protocols is the handling of *hot data* items (see TPC benchmarks and [7]). Note that in the g-2PL protocol, the more a certain data item is requested as for hot data items, more is the performance gain, since the grouping effect is more emphasized when the forward list is longer.

3 System Model for Performance Evaluation

In order to evaluate the performance of g-2PL, a simulation model of the protocol was developed using the *C* programming language. The simulation is a discrete-event simulation using the unit-time approach to advance the simulation clock [8]. The performance of g-2PL is compared with the well known s-2PL protocol which is the industry standard. Thus, in the simulation model, both g-2PL and s-2PL are implemented. We consider a client-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. For instance, this is the case if the network topology has a star configuration with a switch at the center and each node (clients or server) is connected to the switch by a link with the same capacity and length. Thus, communication between any two nodes is achieved in 2 hops through the switch with the same network latency.

Our transaction workload is similar to that in previous work (e.g., [7]). All clients are assumed to be identical and run transactions that have the same statistical profile. The multi-programming 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 (see Table 1). 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 DB server. Each 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 (see Table 1).

As mentioned in the previous section, two-phase locking protocols are deadlock-prone [6]. 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 the case of g-2PL, the forward lists are reordered to ensure that deadlocks are prevented. In the case that such reordering is not possible, the offending transactions are aborted. Each transaction that is aborted is replaced by another transaction.

1.	Number of Servers	1
2.	Number of Clients	varying
3.	Number of hot data items	25
4.	Transaction Execution Pattern	Sequential
5.	Multiprogramming level at clients	1
6.	Number of data items accessed by a transaction	$1 - 5$
7.	Percentage of read accesses	$0 - 100\%$
8.	Network Latency	$100 - 1000$ time units
9.	Computation Time per database operation	$1 - 3$ time units
10.	Idle Time between transactions	$2 - 10$ time units
11.	Window Size (g-2PL only)	varying
12.	Timeout (g-2PL only)	varying

Table 1: Simulation Parameters

Two parameters are of particular importance in the g-2PL implementation: the window size and the timeout value. The window size is measured in the number of requests for a data item that must accumulate in the forward list before the server can dispatch the data item to the first client site on the forward list. In reality, the window size may be tuned for each data item depending on its demand. In our case, since we assume uniform access to all data items, the window parameter for each data item is the same. A large window size will cause more grouping of requests, thus saving the number of rounds and improving the performance. At the same time, a large window size will also cause requests to be delayed while waiting for the requisite number of requests to accumulate. Thus, a timeout parameter is used that bounds the total waiting time; once the timeout expires, the pending requests in the forward list are served. If the timeout expires without any pending requests in the forward list, the server restarts the timer. Note that with a window size of 1, the timeout parameter has no effect on the performance.

Table 1 contains a list of all the experimental parameters and the corresponding range of values of the performance study. Note that parameters 8–10 and 12 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 `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.

4 Simulation Results

In this section, for the sake of brevity only the salient results of the simulation study are presented. More details can be found in [3]. 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, 10000 transactions were generated, requiring a simulation time of upto 25 million time units (upto 20 minutes in real time).

As discussed earlier, 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. Figure 1 contains the average transaction response time plotted against the network latency, for low values of the read probability (p_r = 0.00, 0.25) in a database system with 25 hot data items, 50 clients and each transaction accessing between 1 and 5 data items, using the g-2PL and s-2PL protocols. The window size for the g-2PL protocol is kept at the minimum possible value of 1. Figure 2 contains a plot of the average transaction response time for both protocols with the same database and transaction parameters, but with high values of the read probability (p_r = 0.75, 1.00). Obviously as the network latency is increased, the average transaction response time increases correspondingly. From Figures 1 and 2, it is evident that only when the read probability is 1.00 is the performance of s-2PL better than the g-2PL protocol. The reason for this 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. As the write probability is increased, g-2PL outperforms s-2PL by grouping access requests and saving on the number of rounds. From Figures 1 and 2, the s-2PL response time is upto 25% higher than that with the g-2PL protocol.

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. Figure 3 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 5 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 3, deadlocks are the only cause for transactions to be aborted, i.e., no communication or site failures are assumed. Figures 4 and 6 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 relatively close. However, it can be seen that at both values of p_r, a cross-over in performance occurs and beyond a certain loading, the fraction of transactions aborted in the s-2PL protocol is higher.

The last aspect researched in our performance evaluation was the effect of the window parameter values and the timeout durations on the performance of the g-2PL protocol. Ideally, the window parameter and the timeout duration should be chosen such that the average response time is the minimum possible. The results discussed so far were computed assuming a window size of 1, which renders the timeout duration of no consequence. Note that the window size and the timeout together determine the grouping of requests achieved, and thereby the overall performance. We approached this problem by fixing the window size at a large enough value so that the timeout duration alone determined the grouping. Then the timeout duration was varied and the average transaction response time was studied. As expected, as the timeout duration is increased, the average response time increases as well. However, for the cases studied, there was a range of timeout durations (`0 < timeout < T`) for which the average response time is low and approximately the same with only small variations. Thus it appears that for the cases studied, using a small timeout duration (or a window size of 1) would provide performance close to the best. Further, the difference between the best response time achieved and that with a very small timeout duration was less than 1%. Keeping the timeout duration at a high value and changing the window size produced similar results.

5 Conclusions

The group two-phase locking (g-2PL) protocol was developed for gigabit-networked database systems recognizing that propagation latencies as the bottleneck and that migrating large amounts of data between database servers is not a problem. In order to study the performance of the g-2PL protocol, we have implemented a simulator of a shared nothing, client-server distributed database 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 the strict 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. Interestingly, the g-2PL protocol exhibits worse response time for read-only transactions although one might have expected that both g-2PL and s-2PL protocols would have behaved the same. This shows that the g-2PL protocol fails to fully explore the commutativity of read operations as in the case of s-2PL. Another interesting result is that the window size determining the grouping of requests has almost no influence on the performance of g-2PL and it seems that using a small timeout duration for controlling the grouping of requests is sufficient.

Our future research will investigate inter-transaction caching issues in the context of data shipping protocols, and the ensuing performance trade-offs in the g-2PL protocol.

Acknowledgments

We thank the students who have participated in this ongoing project, especially J. Sen, P. Desai, L. Theophanous and J. Tam. This work has been partially supported by NSF awards IRI-9502091, NCR-9624125 and a Pitt CRDF grant.

References

[1] S. Banerjee and P. K. Chrysanthis. Data Sharing and Recovery in Gigabit-Networked Databases. In *Proc. of the Fourth Intl. Conf. on Computer Communications and Networks: IC3N-95*, pp. 204–211, 1995.

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

[3] S. Banerjee and P. K. Chrysanthis. Network Latency Optimizations in Distributed Database Systems. Technical Report CSD 97-16, University of Pittsburgh, Department of Computer Science, July 1996.

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

[5] 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, 1991.

[6] 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*, 19(11):624–633, 1976.

[7] M. J. Franklin and M. Carey. Client-Server Caching Revisited. In T. Ozsu, U. Dayal, and P. Valduriez, editors, *Distributed Object Management*, pp. 57–78. Morgan Kaufmann Publishers, 1993.

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

[9] L. Kleinrock. The Latency/Bandwidth Tradeoff in Gigabit Networks. *IEEE Communications Magazine*, 30(4):36–40, 1992.

[10] J. C. Mogul and V. N. Padmanabhan. Improving WWW Latency. In *Proc. of the Second World Wide Web Conf., Developer's Day track (http://www.ncsa.edu/SDG/IT94)*, 1994.

[11] R. Ramaswami. Multiwavelength Lightwave Networks for Computer Communication. *IEEE Communications Magazine*, 31(2):78–88, 1993.

[12] 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, 1991.

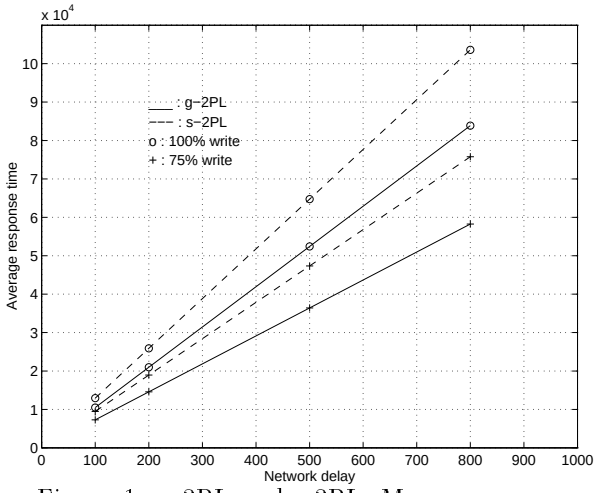

Figure 1: g-2PL and s-2PL: Mean response time vs network latency: 25 data items, 50 clients and high write probability

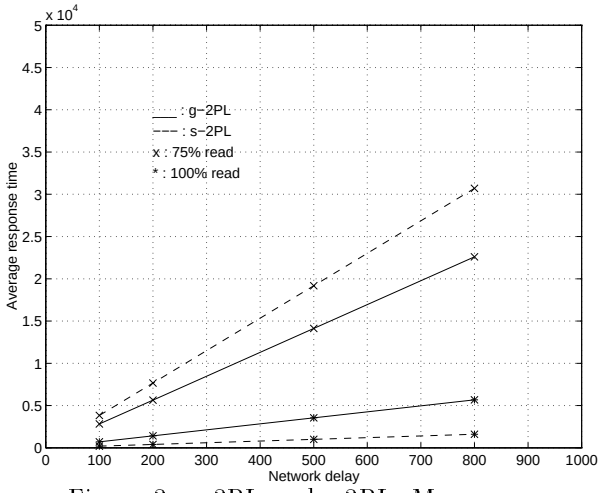

Figure 2: g-2PL and s-2PL: Mean response time vs network latency: 25 data items, 50 clients and high read probability

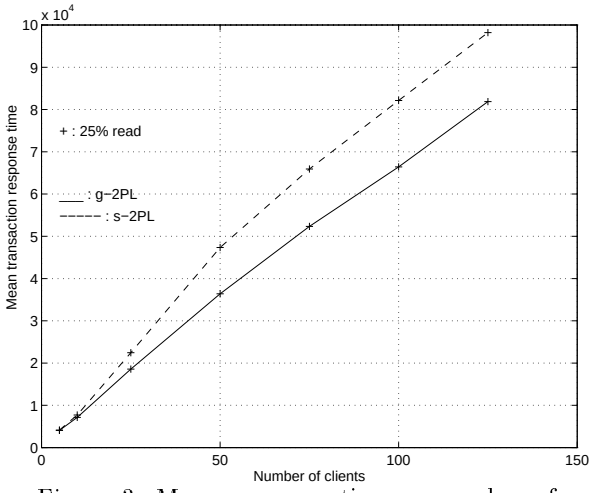

Figure 3: Mean response time vs number of clients: 25 data items, 75% write accesses with a network latency of 500 time units

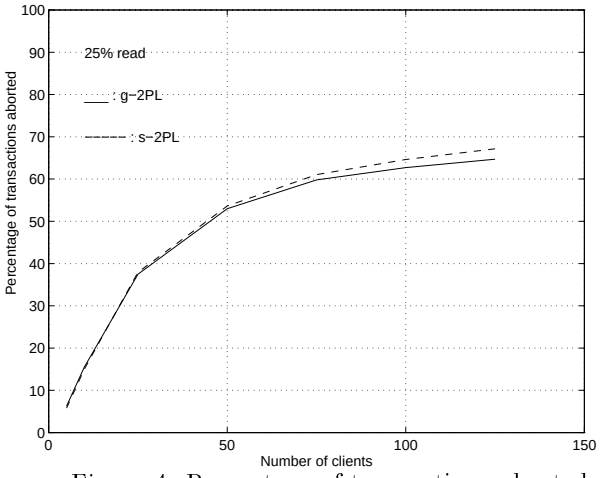

Figure 4: Percentage of transactions aborted vs number of clients: 25 data items, 75% write accesses with a network latency of 500 time units

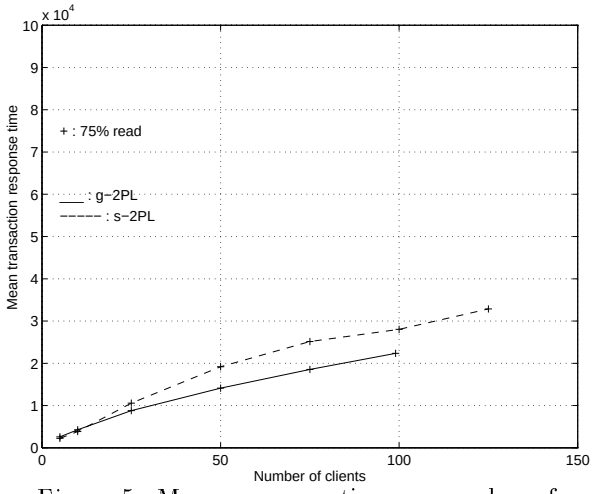

Figure 5: Mean response time vs number of clients: 25 data items, 25% write accesses with a network latency of 500 time units

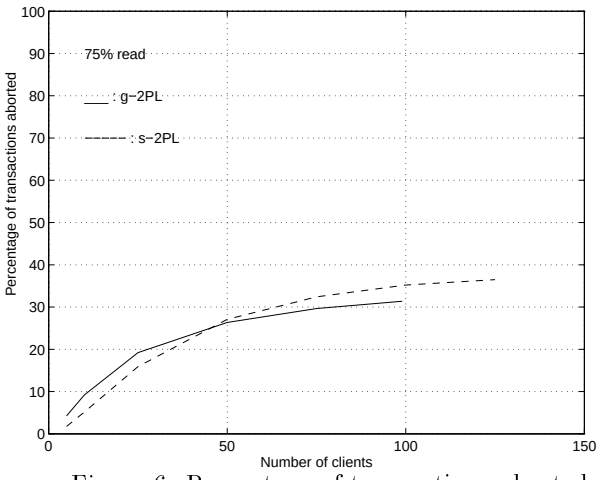

Figure 6: Percentage of transactions aborted vs number of clients: 25 data items, 25% write accesses with a network latency of 500 time units