Concurrency Control in Distributed Object-Oriented Database Systems

Kjetil Nørvåg, Olav Sandstå, and Kjell Bratbergsengen
Department of Computer and Information Science
Norwegian University of Science and Technology
Trondheim, Norway

Abstract

Simulating distributed database systems is inherently difficult, as there are many factors that may influence the results. This includes architectural options as well as workload and data distribution. In this paper we present the `DBsim` simulator and some simulation results. The `DBsim` simulator architecture is extendible, and it is easy to change parameters and configuration. The simulation results in this paper is a comparison of performance and response times for two concurrency control algorithms, timestamp ordering and two-phase locking. The simulations have been run with different number of nodes, network types, data declustering and workloads. The results show that for a mix of small and long transactions, the throughput is significantly higher for a system with a timestamp ordering scheduler than for a system with a two-phase locking scheduler. With only short transactions, the performance of the two schedulers are almost identical. Long transactions are treated more fair by a two-phase locking scheduler, because a timestamp ordering scheduler has a very high abort rate for long transactions.

1 Introduction

With the demand for higher performance and higher availability, computers have moved from centralized to distributed architectures. This introduces new issues in the area of database management. Simulating distributed database systems is inherently difficult, as there are many factors affecting the results. This includes architectural options as well as workload and data distribution. In this paper we present the extendible and easy configurable `DBsim` simulator and some results from a comparison between two concurrency control algorithms, timestamp ordering and two-phase locking.

We have earlier implemented a centralized version of `DBsim`, where standard relational database systems were used as context. From the trends in current database research, we feel that it would be more interesting to simulate object-oriented database systems. While distributed relational database systems usually use query shipping, data shipping is most common in object-oriented database systems. That means, instead of sending the queries to the data, data is sent to the queries. The most popular data granularity is *pages*. This is the easiest to implement, the most common in todays object-oriented DBMS, and also the granularity that gives the best performance [5].

Well-known centralized concurrency control techniques can be extended to solve the problem of concurrency control in distributed databases, but not all concurrency control techniques are suitable for a distributed database. One example is serialization graph testing, which works well in a centralized database system given relative powerful processors compared to I/O speed. But in a distributed environment, keeping the graph updated at all times is prohibitly expensive because of the communication costs.

During the last years, several distributed database systems have been realized. Usually, the concurrency control in these systems has been done by some kind of two-phase locking, but as processor speed increases relative to I/O and communication speed, it is expected that timestamp ordering should be able to compete with two-phase locking in performance. In theory, timestamp ordering scheduling should be capable of good performance in distributed systems. It is deadlock free and avoids much communication for synchronization and lock management.

1.1 Outline of the Paper

The paper is organized as follows. The next section gives a review of related work. In Section 3 we describe the architecture of the distributed version of `DBsim`. In Section 4 we discuss the parameters used in the simulation model, and in Section 5 we discuss the results from the simulations. In Section 6 we discuss possible weaknesses and shortcomings in our model. Finally, in Section 7, we present future work and concludes the paper.

2 Related Work

Much work has been done in studying characteristics of centralized schedulers. An interesting model and simulation results can be found in [1] by Agrawal, Carey and Livny. Less has been done in the area of distributed schedulers. The work that has been done has been mostly theoretical, but some interesting simulation models have been developed and simulated at the University of Wisconsin. In [6] Carey and Livny describes a distributed DBMS model, an extension to their centralized model. Different simulation parameters are examined through simulations. Several papers about concurrency control have also been written by Thomasian et. al. [7, 12].

The most important difference between our approach and the earlier approaches, is that we focus on data-shipping page-server OODBs, while earlier approaches have been done in the context of query-shipping relational database systems. Also, inter-operation and inter-transaction times are expected to be much smaller in this kind of system.

3 The Architecture of `DBsim`

In this section, we present the architecture of the simulator. Each of the main modules will be described. Especially, we will focus on the parts which are particularly important in the distributed model.

In addition to simulate and compare schedulers, one of the main goals in the development of the `DBsim` simulator was that it should be useful as a framework for simulation of schedulers and easy to extend with new schedulers. The `DBsim` architecture is object oriented, all the major components are implemented as classes in C++. The program consists of a collection of cooperating objects, the most important are the event controller (the main loop), transaction manager (TM), scheduler, data manager (DM) and the book keeper. Extending the simulator with new scheduler strategies is easy, e.g., if someone wants to test out a new scheduler, it can be implemented as a subtype of the generic base scheduler class defined in `DBsim`. The generic scheduler has defined the necessary methods to cooperate with the transaction and data managers.

The simulation is event driven, and each transaction can be thought of as a thread. In the main loop an event from the event queue is picked and executed. Events in the queue consists of an event type and the time for the event to be executed. If the event is a TM-event, the TM is called, and if the event is a DM-event, the DM is called. Possible reasons for events could be a transaction requesting an operation, or the data manager has finished a read or write operation on disk.

3.1 From Centralized to Distributed

Compared to a centralized database system a distributed database system is much more complex. We had to augment the simulation model used in the centralized simulator with flexibility to simulate the most important aspects of a distributed database system. We introduced three new concepts to the distributed simulation model:

1. Number of sites that the database is distributed to.
2. Locality of data.
3. Type of network.

The main architecture for our simulator can be seen in Figure 1, which shows the simulator as a collection of cooperating objects. For each simulated node we have one data manager object, and one scheduler object. In our model we

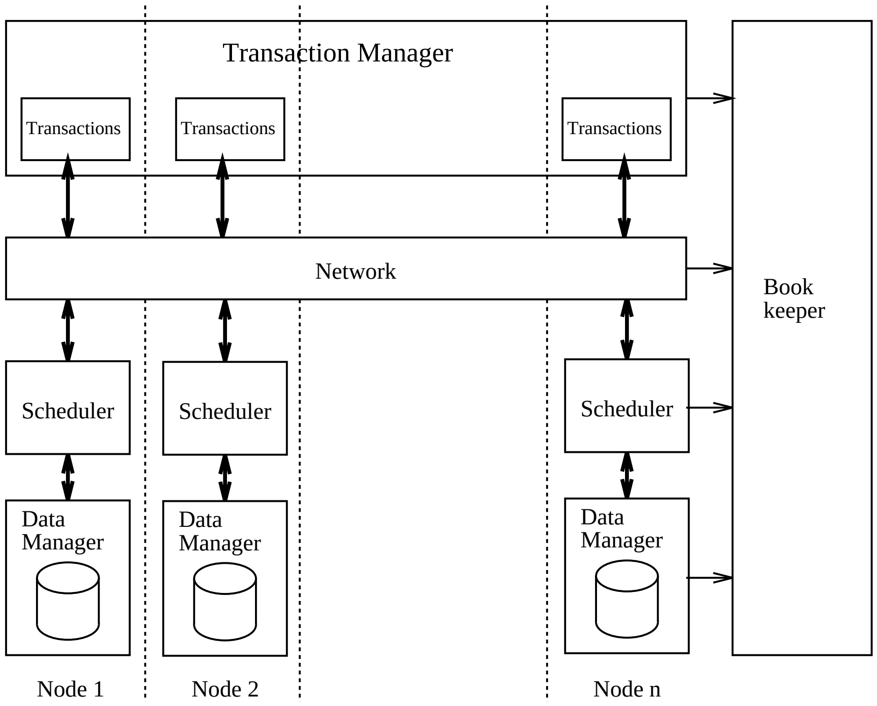

Figure 1: The architecture of the simulator.

have only one global transaction manager object. This object is responsible for creating transactions and operations to the underlying scheduler and data manager objects. A book keeper object is used to collect statistics about the ongoing activities.

3.2 Transaction Manager

The Transaction Manager (TM) is responsible for simulating the transactions that generate operations to the schedulers. It is implemented as one module that contains information about all active transactions.

Every transaction is assigned to one of the nodes in the network when it is created. In this way, although we only have one TM object, we simulate that every node has one transaction manager, and every transaction has one originating node.

In the simulation model we use the number of concurrent transactions as the main criteria for load on the system. In every simulation we have a fixed number of active transactions. This is a good approximation to a system with constant load. In [3], this is called *multiprogramming level* (MPL). Every time a transaction finishes, a new one is started (after some random delay).

Address Space

The database contains a fixed number of data elements. Each data element is addressed by a number in the interval [0, *max*]. The data elements are declustered on all (or some of) the nodes in the network. How the data elements are assigned to the nodes is specified in the simulator's configuration file.

Operations

When a transaction is created it is given “a life on its own”, and starts to generate operations to the schedulers. Each operation is either a Read, Write, Commit or Abort operation. If the operation is a Read or Write, a data element has to be accessed. One of the main advantages with a distributed scheduler is the possibility to store data at the node that most frequently accesses it. Because of this, the transaction selects the data element in most cases to be a data element stored on the same node as the node where the transaction originated. This locality of the data accesses can be configured individually for each node.

Every time an operation finishes successfully, the transaction, after a short delay, generates a new operation or it decides to end the transaction by sending a commit or abort operation.

The previous implementations simulated a mix of long[1] and short transactions. Every time a new transaction was created there was a certain probability that is was a long transaction. The problem with long transactions is that they stay for a long time in the system and occupy more resources. This results in frequent conflicts with other transactions, and often results in aborts of the long transactions. For heavy loaded systems, the long transactions very often get aborted several times before they manage to finish. Results from our simulations with the centralized version of the simulator, showed that even if the system is started with very few long transactions, due to aborting and restarting of long transactions, when ending the simulation there is a much larger amount of long active transactions than short active transactions. To avoid this situation, we have changed the simulation model in a way that a fixed percentage of all active transactions are long. This makes balancing the resource usage between long and short transactions more manageable.

3.3 Network Models

In a distributed database system, the communication cost can contribute much to the cost of retrieving data items. Therefore, it is of vital importance to model this as close to reality as possible.

Computers can be connected in many ways. Of importance here is both the network topology, bandwidth of the network and distance. To avoid a too complex model with too many parameters, we differ between three classes of networks: Cluster, LAN and WAN.

While the size of data items is not of vital importance in the centralized version of the simulator, it is in the distributed version. We have chosen to simulate a page shipping database system with a page size of 4KB.

A cluster in this context is computers connected by a very high-speed network, e.g. ATM. We assume there is point-to-point connection between the computers.

A LAN is a network that connects computers within a building. In this model we assume a broadcasting LAN. Here the delay is the time it takes to send already scheduled packets to the net, plus a constant time it will take to send and receive this packet.

A WAN is a network with longer distances between the computers, and less bandwidth available for each application. The delay is calculated the same way as for LAN, but we expect the time to send and receive a packet to be much larger, due to less bandwidth available. This is a quite naive way to model a WAN, as modeling a generic WAN is very difficult, with many parameters affecting the results. The results we get from the simulation with this model should therefore be taken with a grain of salt.

One of the problems here is: how much time will sending and receiving data packets typically take? Based on Gray and Reuter [9], and measuring the networks in our department, we have used 1 ms for clusters, 5 ms for LANs and 840 ms for WANs. In our model, this time also includes overhead and time used for interprocess communication. The difference between LAN and clusters is small, this is because the overhead before and after transmitting is the same in both models, and takes considerable part of the time.

The results reported in this paper are all from simulations with a LAN.

[1]The term *long transaction* in this context is not a transaction of long duration as is common in design systems, but a transaction with a larger number of operations than the typical small transactions (in our model 14-100 operations vs. 2-8 operations).

3.4 Scheduler

The scheduler module in this system is a *black box*, it communicates with the other modules only via a well defined interface. This makes it easy to replace it with schedulers that use other techniques. In the distributed version of the simulator, two-phase locking and timestamp ordering schedulers are implemented.

The scheduler receives an operation from the transaction manager, and processes it according to its scheduling technique. When the scheduler decides that an operation can be sent to the data manager, the data manager is called. When the operation has completed, the scheduler will be notified by a call from the data manager.

The following distributed schedulers are implemented:

- Strict Two-Phase Locking Scheduler:

  A nice feature with non-replicated distributed databases is that each local scheduler can schedule the data accesses as if it was a centralized scheduler. But of course there are also some problems that are more difficult to solve in the distributed context than for a centralized scheduler. For the 2PL scheduler the main problem which has to be solved is *deadlock*.

- Strict Timestamp Ordering Scheduler:

  We have implemented a *strict* TO scheduler. Although deadlocks are no problem here, we have another “global problem”. Assigning monotonous increasing unique timestamps to transactions. In a real implementation this could be done by concatenating a local timestamp counter with the node number. Keeping the clocks synchronized is not trivial, but one solution to this problem is discussed by Lomet in [10].

3.5 Data Manager

The data manager simulates the physical database storage manager. Each node has a data manager, which operates independently of the other data managers in the system. We have used a very simple model, where the data manager only consists of a simulated buffer cache and a disk.

Time delays are used to simulate operations against the data manager. The time needed to execute a disk operation is the sum of the time that operation has to wait for the disk to become idle (the time needed to execute previous operations submitted to the disk), and the time it takes to read or write the page. We have chosen a very simple algorithm for disk scheduling: *first-come-first-served* (or *first-in-first-out*). A read operation from the cache takes constant time, and can be served immediately.

Logging is not implemented separately. The time and resources needed for logging is considered to be included in the time needed to do the write operations. However, when an abort occur, we write back the before images.

3.6 Commit Protocol

The centralized version of the simulator used a very simple commit protocol. This can be done because there is only one scheduler. In a distributed database, it is common that more than one scheduler participate in executing a transaction. Because of this, a distributed commit protocol has to be used, to make sure that all participating schedulers reach the same result. Either all perform the commit, or all have to abort. Two well-known protocols are *two-phase commit* (2PC) and *three-phase commit* [11]. We have employed 2PC, which is the protocol used by most commercially available distributed database systems.

3.7 Book Keeper

The book keeper is used for collecting statistics. Among other things, it counts the number of transactions completed and aborted, whether they are short or long, and average execution time. It also collects statistics about active transactions (transactions that are not finished) at the time the simulation ends.

Parameter	min	max
# operations in short transactions	2	8
# operations in long transactions	14	100
# operations in a burst	3	5
Time between transactions	10 ms	100 ms
Time between operation requests	1 ms	10 ms
Time between operations in a burst	1 ms	3 ms
Time to perform a disk operation	8 ms	16 ms
Restart delay	500 ms	1500 ms

Table 1: Min and max values of interval parameters.

Parameter	Value
Number of transactions	20000
Size of address space, # of resource units	20000
Hot spot size, # of resource units	2000
Hot spot probability	50%
Short transaction probability	80%
Abort probability	0.1%
Read probability	80%
Burst probability	20%
Block size	4 KB
Block transfer time over the network (LAN)	5 ms

Table 2: Parameter values.

4 Simulation Parameters

The performance in real database systems is determined by several parameters. Some of them are determined by the hardware and operating system architecture, some are set by the administrator, and others are results of the characteristics of the transactions done on the system. If we want the simulation results to have value in real life, these parameters have to be a part of the model. In this section we will briefly present the most important parameters that are part of the simulation model. Determining reasonable values of the parameters are very important, as the final result is highly dependent of these values. Our parameters are mainly taken from [8] and [9], others are determined according to hardware specifications and previous work at our department. The parameters being constant for all simulations are summarized in Table 1 and 2.

It should also be noticed that all transactions are *strict*. Strict execution is dominant in real systems, both because of user functionality and recovery reasons. Strict execution implies that all resources are held until commit/abort.

Number of Nodes. Number of nodes in the system we simulate. These nodes are connected by a network.

Number of Transactions. The number of transactions to complete after the initial warm-up phase. The reason for having a warm-up phase, is to make sure that the start-up phase for the system we simulates, is not taken into account when we start collecting statistics about the simulation. In our simulations we have used a warm-up phase consisting of 2000 transactions before we start collecting data.

Size of Address Space. Number of elements (pages) in the address space is important. Agrawal et. al. [1] argues for setting this to a low value, to increase the number of conflicts. In our simulations we have set the address space to 20000 elements.

Data Declustering. Distribution of the data elements to nodes, the percentage of the database elements in the database system located at a particular node.

Data Locality. The probability that a transaction access a data element located on the same node as the transaction.

Non-Local Data Access. When a transaction in the system is accessing a data element not located at its home node, this is the probability that the remote access will access data at a particular node.

Hot Spot Probability . The probability of an operation to address the hot spot part of the address space. The hot spot area in our model is the first 10% of the address space at each node.

Multi Programming Level. Number of concurrently executing transactions.

Transaction Distribution. The probability that a new transaction in the system is started on a particular node.

Short Transaction Probability. The probability of a transaction being *short*. The rest of the transactions are long transactions.

Abort Probability. The probability of a transaction requesting abort before committing. This probability is the same for both long and short transactions.

Read Probability. The probability of a data access operation being a read operation. In our simulations, we have used a value of 80%. This gives a write probability of 20%.

Burst Probability. The probability of a transaction to ask for operations in a burst. Time between operations in a burst is shorter than normal. The length of short and long transaction, the time between transactions, time between operation requests from a transaction and number of operations in a burst is drawn from an uniform distribution with parameters as shown in Table 1.

Restart Delay. When using a timestamp ordering scheduler, transactions might get aborted because of conflicts. If restarted immediately, the probability for the same conflict to occur is quite high. To avoid this problem, the restart of the transaction is delayed a certain amount of time. This delay is a value drawn from a uniform distribution, multiplied by the number of retries.

Network. What kind of network to simulate. It can be of three types, cluster, LAN or WAN, as described in Section 3.3.

Scheduler Type. Scheduler used for concurrency control. In the current version, this has to be either two-phase locking or timestamp ordering.

Disk Operation Time. This time interval gives the time for doing an access to the disk. This time is drawn from a uniform distribution shown in Table 1.

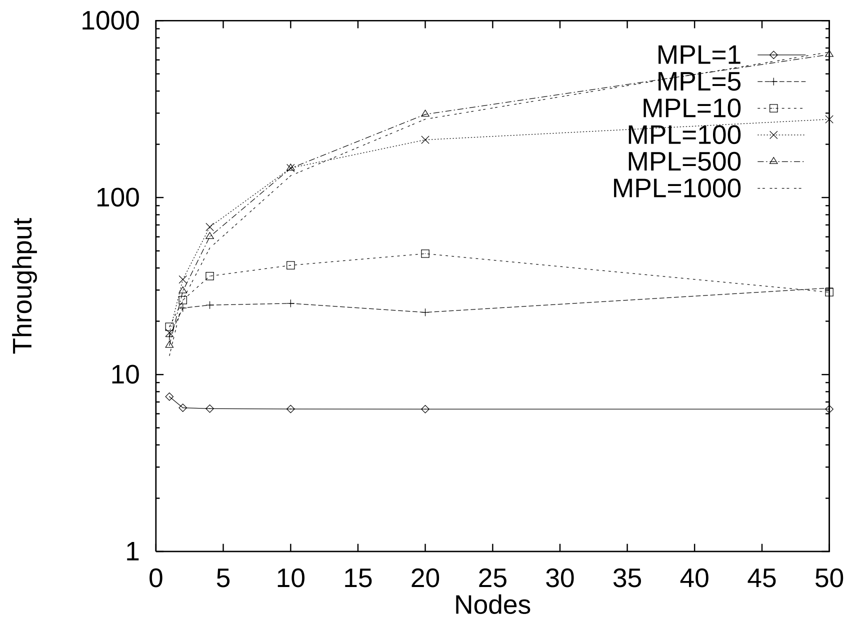

Figure 2: Throughput vs. number of nodes and MPL, TO scheduler.

5 Simulations and Results

The results of the simulations should make us able to say something about the schedulers performance, relative to each other. If these results should have any relevance in "real life", the mix of operations given to the schedulers should be as close to those in real life as possible. Different systems have different characteristics, and to reflect this, our simulations are done with a mix of different characteristics.

Our simulations are based on a constant load. The MPL during one simulation is constant. We have used a mix of short and long transactions. At all times, 20% of the load on the system comes from long transactions.

When one transaction finishes, a new one is started. The number of finished transactions in each simulation run is 20000, with an additional warm-up phase of 2000 transactions finished before we start collecting statistics. We have tried to observe the effect of distribution on two-phase schedulers and timestamp ordering schedulers. We measured throughput with different number of nodes and MPL, with different data placement, and different types of networks.

5.1 Throughput vs. Number of Nodes and MPL

Distributed DBMS are used both to increase availability and performance. The performance should increase with increasing number of nodes. We have simulated with 1, 2, 4, 10, 20 and 50 nodes, and for all of these node configurations with a MPL of 1, 5, 10, 100, 500 and 1000. The results are shown in Figures 2, 3, and 4. Here we assume that all nodes have an equal amount of the data in the database, and the transaction load is balanced. That means that on average the same number of transaction are started on each node. We also assume that the probability of the data being accessed to be located at the home node of the transaction to be 80%. If not on the home node, it is located on a remote node, each node having the same probability. The network model used in this simulation is a LAN.

As can be seen from Figure 2 and 3, the throughput increases with increasing number of nodes for both timestamp ordering and two-phase schedulers. This is as expected.

A more interesting result is that the throughput for the TO scheduler is higher than 2PL scheduler. This was not actually expected. In the comparison done with the centralized version of the simulator, the 2PL scheduler performed better than the TO scheduler in most cases. In a real implementation of a distributed scheduler it would have been

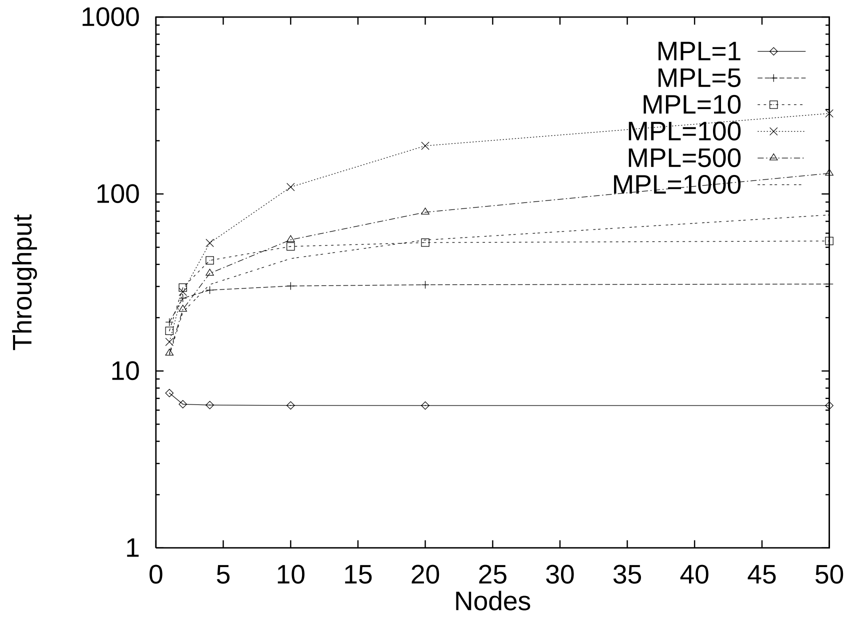

Figure 3: Throughput vs. number of nodes and MPL, 2PL scheduler.

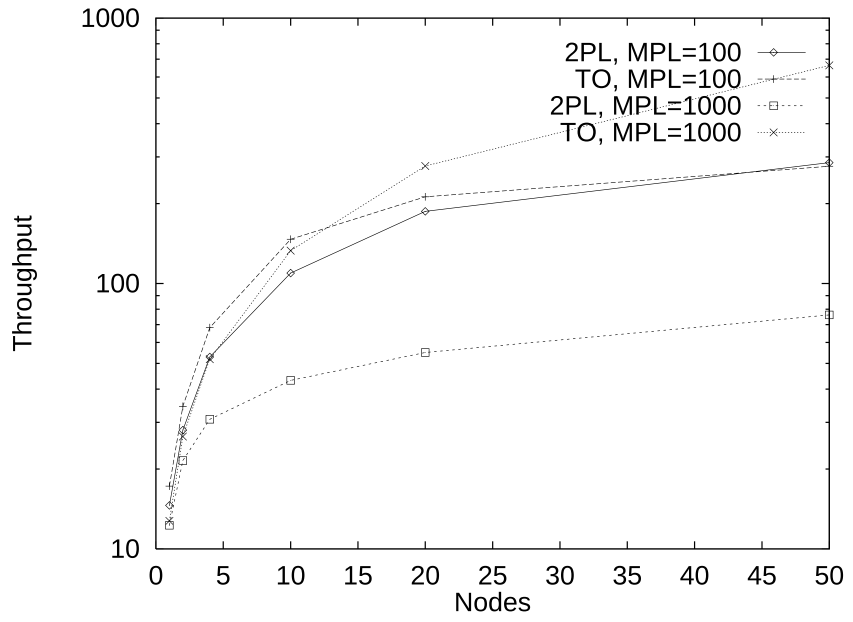

Figure 4: Throughput vs. number of nodes and MPL, the two schedulers compared.

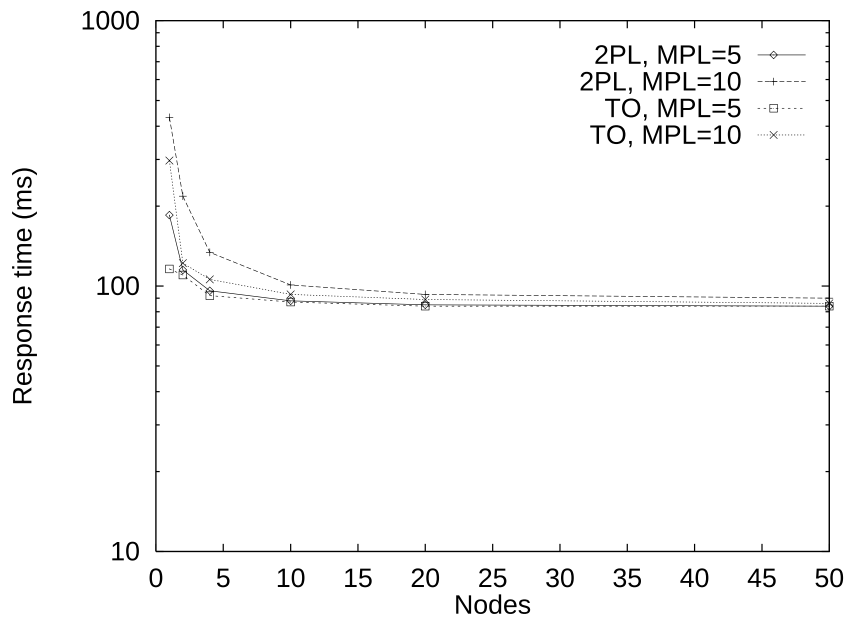

Figure 5: Response times for short transactions, low MPL (giving low data contention).

more likely to experience this result, because the 2PL has an overhead due to the global deadlock detection. But in our implementation, the global deadlock checking is done without any delay on the participating transactions. Why we experience this result will be discussed in Section 5.3.

For the TO scheduler (Figure 2) we note that the curves for a MPL of 500 and MPL of 1000 are almost identical. We have here reached a limit for maximal throughput in the system with the available resources, where the main bottleneck is the disks.

For the 2PL scheduler (Figure 3) we reach the maximal throughput for a much lower load. The figure shows that we get the maximal throughput near 100 active transactions. For 500 and 1000 active transactions, the throughput is much lower. The reason for this is discussed in Section 5.3.

With only one transaction in the system, the throughput is expected to decrease when we add several nodes. This can be verified by examining the figures. When we only have one active transaction, the line make a dip when we introduce more than one node in the network. The reason is that some of the data accessed have to go through the network.

Another comment we want to make to these figures, is that with more nodes, and a network with high capacity between them, the databases get a very good increase in performance compared to a centralized database. The main reason is higher disk bandwidth because of more disks. This, of course, requires that data are evenly accessed and that most accesses goes to the local disk without going through the network. Given these characteristics, it seems like a distributed database consisting of rather inexpensive nodes based on off-the-shelf technology very well can compete with more expensive parallel database machines.

5.2 Response Times

In Figure 5 and Figure 6, we present the response times for the two schedulers. The results are taken from the same simulations as those presented in the previous subsection. We have only used the short transactions when making these curves. The reason is that it is usually the small transactions that have the most stringent requirements regarding the response time.

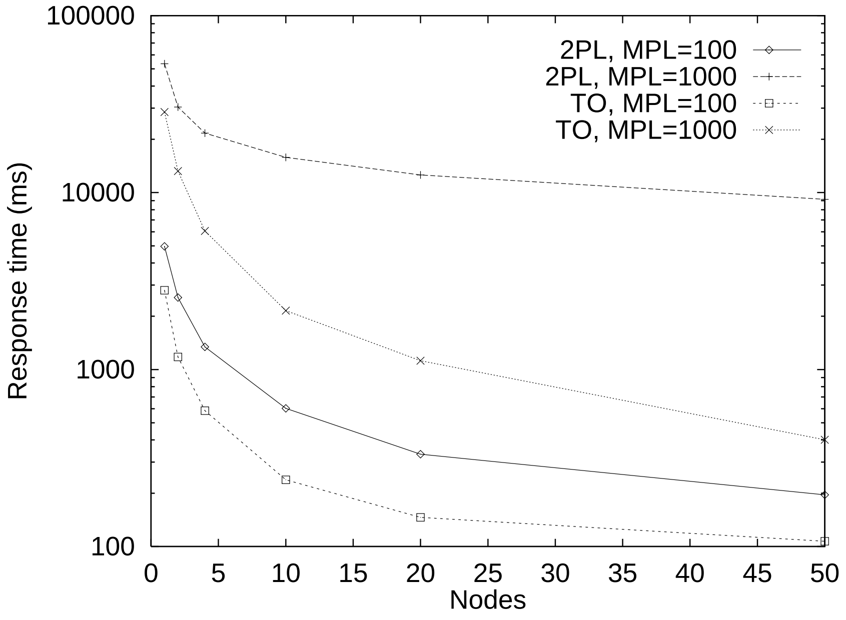

Figure 6: Response times for short transactions, high MPL (giving high contention).

If we compare the two figures we reach the same conclusion as in the previous subsection: The TO scheduler performs better than the 2PL scheduler for short transactions.

5.3 Abort Frequency

We are now able to answer the question why the TO has higher throughput than the 2PL scheduler. The length of a transaction influence very much on the abort probabilities. For short transactions, the abort probability is rather small. For long transaction, this is not the case. Especially if we have a high number of parallel transactions, we get a lot of conflicting operations, and many aborts. What is important to notice, is that the TO produce more aborts than the 2PL scheduler. This is as expected, since the TO doesn't handle conflicts as well as the 2PL scheduler. The 2PL is more likely to let transactions wait for the resource, while the TO aborts more often when a conflicting operation occurs.

Then *why does the TO scheduler perform so much better than the 2PL scheduler?* The main reason, is that we have a mix of long and short transactions. The long transactions are the reason why the 2PL performs worse than the TO. There are two reasons for this:

1. With many active transactions, there will be a lot of conflicts, and with long transactions in the system, they will lock many data elements. The 2PL will let a short transaction wait for the long transaction to finish. This makes a lot of short transactions being locked for long time. This is the main reason why the response time is so much longer for short transaction with the 2PL scheduler, compared to the TO scheduler (see Figure 5 and Figure 6). It is also the reason why the 2PL reaches its maximum throughput earlier than the TO.

2. Another effect of the locking is that long transactions have a higher probability of finishing without being aborted. This means that in average there are more resources occupied for long time by long transactions in the 2PL scheduler than in the TO scheduler. From the simulation logs we noticed that the percentage of the finished transactions that where long was much higher for the 2PL than for the TO.

A conclusion that can be drawn from this is that the 2PL treats long transactions more fair than the TO scheduler. The cost of this is that the maximum throughput is not as good. To check this we ran the same simulations with

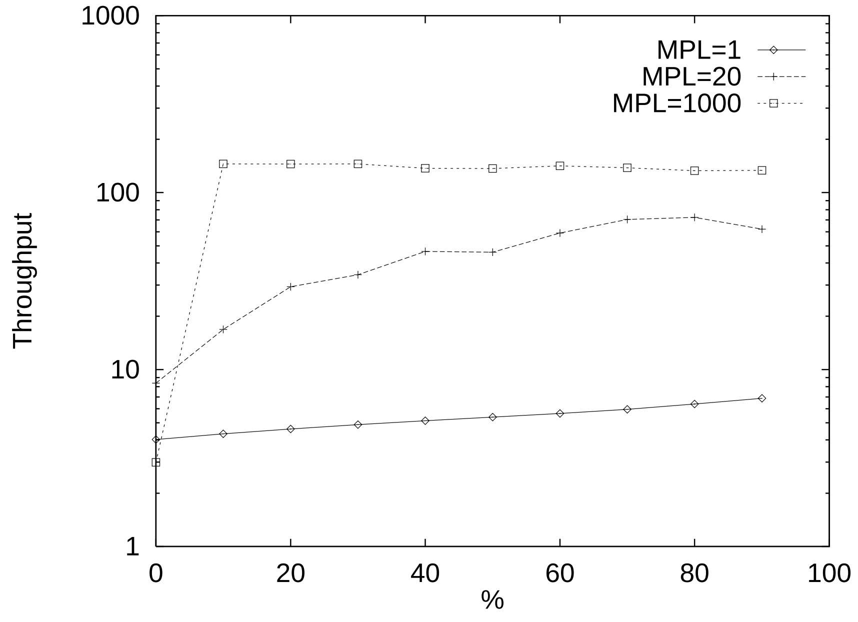

Figure 7: Throughput with changing home node probability, TO.

only short transactions. This time both schedulers had almost comparative performance, as expected. For a mixed load, i.e. long and short transactions, we keep the load ratio constant, 20% long and 80% short transactions. However, the *production ratio* changes with increasing total load. Because the long transactions are more vulnerable than short transactions to data contention under the TO scheduler, the relative production of long transactions is decreasing with increasing MPL.

5.4 Throughput with Different Data Placement

Also important is to measure throughput when we change the probability of accessing data on home node. Here we used the schedulers with a configuration with 10 nodes and a MPL of 1, 20 and 1000, with a LAN network.

First, we saw how the throughput changes with a home node probability ranging from 0% to 100%. This result is presented in Figure 7 and 8. As expected the curves shows that the more of the data accesses that go to local data, the better is the performance of the system.

For the TO simulations, the curves for a MPL of 1 and a MPL of 20 are easy to explain. For a MPL of 1 the throughput only increase slightly when increasing home node probability. The reason is that more and more of the operations can be performed locally and do not have to go across the network. For a MPL of 20 we see a very good effect from increasing the home node probability. The system goes from a state where the network is the bottleneck, to a state where the network is contributing very little to the transaction delay. The curve for a MPL of 1000 might look a bit strange. For 0% home node probability the network is the bottleneck. This is because we simulate the network as a shared medium LAN. For 10% home node probability, the problem has already changed from the network being the bottleneck, to the disks being the bottleneck. From Figure 8, we see that for the 2PL simulation, the results are similar.

5.5 Summary of Simulation Results

The results of the comparison between the timestamp ordering scheduler and the two-phase locking scheduler, can be summarized as follows:

- With a mix of long and short transactions, the TO scheduler have a higher throughput than the 2PL scheduler.

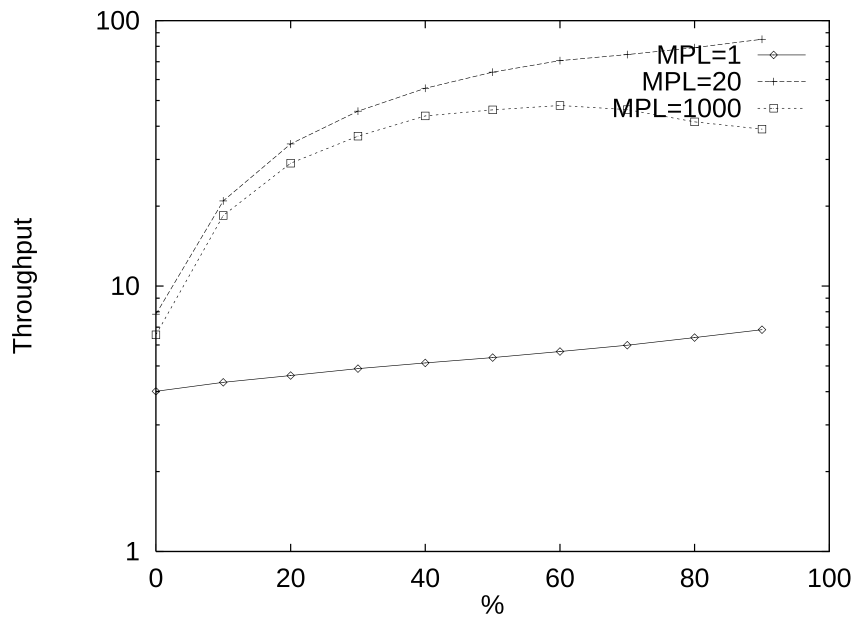

Figure 8: Throughput with changing home node probability, 2PL.

- With only short transaction, the two schedulers performs almost identical.
- The TO scheduler have much higher abort probabilities than the 2PL scheduler.
- The 2PL scheduler is in favor of long transactions, and the number of long transactions that manage to finish successfully is much higher for the 2PL scheduler.
- The network is not the bottleneck for a reasonable load. Only for heavy load and a slow network this severely affects performance.

The main reason for the difference between 2PL and TO, is the way the two schedulers treats long transactions. The 2PL solves conflicts by blocking the transactions, while the TO in many cases aborts the transaction to solve the conflict. This more aggressive strategy is more harmful to long transactions than short transactions. We should also mention that transaction delay is caused by both resource contention and data contention. As 2PL gives more priority to the longer transactions, 2PL would be more vulnerable to the higher levels of data contention, which we have when MPL is high.

6 Weaknesses in the Simulation Model

Performance analysis of any reasonable complex system, real or simulated, is still an art because (among other things): 1) only a small region of the performance space can be explored, 2) it is often difficult to determine the cause of the observed performance behavior, and 3) it is difficult to generalize the results of specific tests for use in predicting performance under other conditions. Also, as is obvious, no model is perfect, and some simplifications have to be done. This is also the case with `DBsim`. However, they should not significantly affect the value of the results, but the results should be interpreted qualitatively rather than quantitatively.

Several parts of the model can be improved. The network models could be more advanced. Also, the very different access patterns in modern and future databases is difficult to predict and simulate. In reality, only a real implementation can give reliable answers.

7 Conclusions and Further Work

We have described `DBsim`, a model and simulator for a distributed page-server based object-oriented database system. The simulator is modular and extendible, and in this paper we have given results from simulations with two different scheduler strategies.

Further work for the `DBsim` simulator includes extensions that could make it more suitable for simulation of algorithms for object-oriented databases. Obviously, much more can be done with both the simulation model and the simulator. This includes adding new schedulers to the system, e.g., other versions of the two-phase locking scheduler, like wound-wait and wait-die. In a real system, replication is used for increased reliability and performance. This could also be integrated into this framework.

More information about the data structure could be put into the model, facilitating the use of more advanced scheduling algorithms. One particularly interesting scheduler strategy, is multi granularity locking, which is a) easy to implement and b) efficient, and able to give a considerable improved performance.

If one wants to model the databases of the future, more modern transaction concepts could be evaluated, such concepts are described in [4] and [2].

8 Acknowledgments

We thank Rune Hjelsvold for contributing to the first version of the centralized database system simulator.

References

[1] R. Agrawal, M. J. Carey, and M. Livny. Concurrency Control Performance Modeling: Alternatives and Implications. *ACM Transactions on Database Systems*, 12(4), 1987.

[2] N. S. Barghouti and G. E. Kaiser. Concurrency Control in Advanced Database Applications. *ACM Computer Surveys*, 23(3), 1991.

[3] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley Publishing Company Inc., 1987.

[4] A. Biliris, S. Dar, N. Gehani, H. Jagadish, and K. Ramamritham. A Flexible Transaction Facility for an Object-Oriented Database. Technical report, AT&T Bell Labs, 1992.

[5] M. J. Carey, M. J. Franklin, and M. Zaharioudakis. Fine Grained Sharing in a Page Server OODBMS. In *Proceedings of the 1994 ACM SIGMOD*, 1994.

[6] M. J. Carey and M. Livny. Parallelism and Concurrency Control Performance in Distributed Database Machines. In *Proceedings of the 1989 ACM SIGMOD*, 1989.

[7] P. A. Franaszek, J. T. Robinson, and A. Thomasian. Concurrency Control for High Contention Environments. *ACM Transactions on Database Systems*, 17(2), 1992.

[8] M. Franklin. *Caching and Memory Management in Client-Server Database Systems*. PhD thesis, University of Wisconsin-Madison, 1993.

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

[10] D. B. Lomet. Consistent Timestamping for Transactions in Distributed Systems. Technical Report CRL 90/3, Digital Equipment Corporation, Cambridge Research Lab, 1990.

[11] M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice-Hall, 1991.

[12] A. Thomasian. Performance Limits of Two-Phase Locking. In *Proceedings of the IEEE International Conference on Data Engineering*, 1991.