

Database Concurrency Control on a Shared-Nothing Architecture Using Speculative Lock Modes.

August Climent, Miquel Bertran, Miquel Nicolau

Enginyeria i Arquitectura La Salle
Ramon Llull University, Barcelona.
{augc,miqbe,miqueln}@salleURL.edu

Abstract. This paper presents a new concurrency control algorithm for parallel database systems with a Shared-Nothing architecture. The new algorithm, Speculative Two Phase Locking sp-2pl, is an extension of Strict Two Phase Locking s-2pl. Other concurrency control algorithms outperform s-2pl under certain system conditions, but in general s-2pl gives better results. Our proposal, sp-2pl, permits the advancement of some operations but giving always a serializable schedule. This situation is very useful, because the network overhead of our proposal has minimum impact on a system based on a fast interconnection network or on a bus. Our performance studies also show that the sp-2pl gives a better behaviour than the s-2pl, maintaining the robustness of s-2pl.

1 Introduction

A parallel database attempts to achieve high performance by parallelism giving high performance even for complex queries with very large volumes of data [10]. One of the most important problems in these systems is that in order to preserve data consistency, the transactions tend to set locks on data. Then, the system concurrency control algorithms try to improve system throughput, maintaining information consistency [8]. There are three kinds of concurrency control algorithms CCA: lock based [1], time based [3] and optimistic [7]. These algorithms may be pessimistic or optimistic.

The pessimistic algorithms assume that there can be a conflict, and validate the compatibility of operations at operation requests. The optimistic algorithms do not validate operations before their execution, but at the end of transactions validate all the operations. The most used Concurrency Control Algorithm CCA is s-2pl or Strict Two Phase Locking. This CCA is pessimistic and it is based on locking the information when there is an outstanding operation on it.

In this paper, we present the new algorithm sp-2pl, based on s-2pl but extending the Compatibility Lock Modes CLM [9] which relaxes consistency requirements, obtaining a more efficient CCA. We also study the effect of different workloads in order to compare sp-2pl with s-2pl and Optimistic Two Phase Locking o-2pl.

Previous work is discussed in section 2. The simulation model and input workloads are discussed in section 3. In section 4 we will discuss the different variations of the s-2pl and explain our proposal (sp-2pl). In section 5, we present a variety of simulation results. Section 6 presents the algorithm benefits. Our conclusions and future research directions are described in section 7.

2 Previous Work

The s-2pl, Distributed Time Ordering d-to and o-2pl are the most studied and used CCAs. In this section we will summarize their behaviour.

2.1 s-2pl algorithm

In this algorithm, every read or write operation has a lock status. If the operation is a write, it has always a write lock and if the operation is a read, the lock status can be either a read or a write lock.

The transactions are divided into two phases: a growing phase, where locks and the information are set, and a shrinking phase, where locks are released. When the system has to set a lock, it must validate whether this is possible. This validation is done following the Compatibility Lock Modes CLM [9]. So, if there is a conflict, the operation waits until the conflict disappears, and if the operation is compatible, the lock is granted and the operation can be executed.

When a commit or an abort is requested, the system validates the compatibility of all the operations that are waiting. If there are waits pending, due to incompatible operations, it would be possible to reach a deadlock situation. So, the system must detect these situations using Wait For Graphs WFG and solve them. An additional complexity is that the deadlock may be a distributed one, and all the sites must update their local WFGs in order to construct a Global Wait For Graph GWFG.

2.2 d-to algorithm

The d-to algorithm is based on Time Stamps TS [2]. Therefore, the system detects a compatible or non-compatible operation based on the operation TS. If an operation can not be done, the transaction is aborted. Since there are no WFGs, distributed deadlocks are not possible.

2.3 o-2pl algorithm

The o-2pl algorithm assumes that conflicts between transactions are unusual. Therefore, whenever an operation must be done, the system assumes compatibility and executes the operation. When a commit is received the system validates whether all the operations assumed to be compatible are in fact compatible. In other words, the o-2pl delays the validation phase until the end of the transaction and before the write phase.

3 Simulation Model

As mentioned in Section 1, we have developed a single Parallel Database model for studying a variety of CCAs and performance trade-offs.

The clients send their requests grouped in transactions to the processors and they return the information to the client, and in order to begin a new transaction, the previous one has to be finished. Every processor has three subsystems: A Transaction Manager TM, a Concurrency Control Manager CCM and a Data Processor DP.

1. **Transaction Manager.** The TM organizes the transaction execution of operations that receives from a Client or other TMs. In our model, a Client must always send the operations to the same site.

When one TM wishes to send a message or operation to another site, it must use the network. This can happen when a TM is asked to serve information which it does not have, then it must ask another TM for that information. In this case, it is sure that it will receive the information with one Ack status later, and the TM must send the response to the Client. But if a TM asks for information and receives a NotAck, it must send a Restart or an Abort operation to all the sites participating in the transaction.

2. **Concurrency Control Manager.** The CCM is the subsystem that coordinates the execution of the operations received from the TM according to the CCA used. In CCAs that need locks, the CCM uses lock structures, and in those based on TS it must save Clocks and Locks based on TS. If the CCM receives an operation from the TM, it must decide first whether it is possible to do it or not. If the operation can be done, the CCM sends the operation to the DP and waits for the response. But if the CCM receives an operation that can not be done because there is a compatibility problem, it will produce a different reaction depending on the CCA used.

3. **Data Processor.** The DP receives operations from the CCM for their execution. We only consider two kinds of operations, read and write. When the DP receives a read, it reads the information and gives the result to the CCM with an Ack. When a write operation is received, the DP modifies the information in a temporary copy waiting for the commit, and answers with an Ack to the CCM.

The DP can also receive two termination operations, commit and abort. When it receives a commit, the DP must update all the transaction writes onto disk, and if it receives an Abort, the DP must leave the disk information as it was before the transaction began.

4 The Algorithm

Previous studies have showed that s-2pl is the best CCA in systems with a medium or high level of concurrency. As we know that o-to is a good algorithm in systems with a low level of contention, we extend s-2pl in an optimistic manner.

Since the algorithm assumes, under certain conditions, that some operations are compatible, we say that it is a Speculative algorithm.

The algorithm s-2pl uses the basic Compatibility Lock Modes, CLM. Since we extend the CLM, it makes it possible to improve the lock compatibility mechanism.

As the s-2pl, the sp-2pl can also deadlock. In order to detect this situation, the different TMs have to detect it. The model used is the following: Local Deadlocks are checked whenever a transaction blocks. Global Deadlock detection is handled by a snoop process, which periodically requests the Wait For Graph WFG of every site and then checks for and resolves any Global Deadlock.

4.1 Transaction Manager

The TM receives operations from the Client in order to coordinate their execution. It also receives messages from its CCM and from the other TMs. Messages from its CCM are responses from local operations. When it receives a message from the Client, the operation can be: Begin, Commit, Abort, Read and Write. The associated operations of the TM are the following.

1. Begin. It answers with an Acknowledge to the Client.
2. Commit. It executes a 2PC protocol [6] in order to coordinate the operation.
3. Abort. It propagates the operation to all the sites that have participated.
4. Read. If the lock associated with the Read is a Read Lock, the TM only sends the operation to one site. If the lock associated with the Read is a Write Lock and the information is not replicated, the TM proceeds as in a Read Lock. But if the information is replicated, the TM begins a two phase protocol, where all the sites that have a replica are involved.
5. Write. If the information is not replicated, the operation is sent to the site that has the information. If the information is replicated, the TM begins a 2PC protocol sending first a Lock message to all the sites that have a replica.

When the TM receives a message from another TM and has no associated information, it is interpreted as an operation redirected from another TM, and it is passed to the local CCM. But if it has associated information, it is treated as if it were received from its local CCM, and the operations that can be received are: Read, Write, Lock, VoteCommit, Commit or Abort. The response of the TM are the following:

1. Read. It sends an Ack message to the Client.
2. Write. It must wait until all the participants associated with the Write operation respond. Then it sends an answer to the Client.
3. Lock. The lock is always a Write Lock. This lock can be assigned to either the Read or Write operations. If the operation is a Write, the TM verifies that all the sites have answered, and then sends the operation to all the sites that are involved in the operation. But if the operation is a Read, it only sends the operation to one site.

4. **VoteCommit.** If all the sites have answered, it decides the GlobalVote. The GlobalVote is taken by the TM coordinator depending on the set of votes received from all the participants in the protocol.
5. **Commit or Abort.** If all the participant sites have answered, it sends an Ack operation to the Client. Otherwise, the TM waits for more Commit or Abort responses.

4.2 Concurrency Control Manager

Our improvements in the sp-2pl are reflected mainly in this module. Table 1 shows our Extended Compatibility Lock Modes, ECLM. With these new conditions, it is possible to improve the lock compatibility mechanism.

When the CCM receives a lock request, it decides its response depending on the kind of lock and operation requested Op_r and the lock and operation already granted Op_g . As Table 1 shows, there are three different responses: Compatible (OK), NotCompatible (NOT) and Speculative (SP). If the operation is Compatible, the CCM sends the operation to the DP. If the operation is NotCompatible, the CCM adds this operation as a Pending Operation, because it can not be done. If the operation is Speculative, it means that the operation can be done, but before committing the transaction an additional validation must be done.

Table 1. Extended Compatibility Lock Modes

Op_g Granted	Op_r Requested		
	$r_x(rl)$	$r_x(wl)$	$w_x(wl)$
$r_x(rl)$	OK	OK	OK
$r_x(wl)$	OK	NOT	NOT
$w_x(wl)$	SP	SP	NOT

The CCM receives messages from the TM and from the DP. When it receives a message from the TM, its response depends on the type of message, as follows:

1. **Read or Write.** The CCM validates their compatibility. If the operation is Compatible, the CCM sends the operation to the DP. If the operation is NotCompatible, the CCM adds this operation into the LWFG structure and checks whether the system deadlocks or not. If the operation is Speculative, this means that the operation can be executed but, before completing the transaction, an additional validation must be done. So, the CCM sends the operation to its DP, and later, in the 2PC protocol, this pending validation will be done.
2. **Lock.** The CCM validates the compatibility of the operation, and answers the TM in the same way as when it receives a Read or a Write operation.

3. **VoteCommit.** The CCM should return its vote to the TM. If it has no speculative operations **CommitTran** is returned, but if it has speculative operations, additional validation must be done. If all speculative operations are compatible, **VoteCommit** is returned, but **VoteAbort** if one or more speculative operations are no compatible.

A speculative operation is compatible if all transactions that produced the speculative behaviour have already been finished with a commit operation. Otherwise it is not compatible. If the transaction associated with the granted operation has not yet finished, the CCM adds **VoteCommit** to its LWFG as a pending operation, because another transaction must finish before.

4. **Commit or Abort.** The CCM sends the message to the DP.

When the CCM receives an operation from the DP it can be a Read, Write, Commit or Abort. Its response depends on the type of operation. If it is a Read or Write, the CCM sends the result to its TM. When the operation is a Commit or Abort, the CCM sends an Ack to its TM. Later on, it looks for pending and compatible operations.

5 Performance Evaluation

We have modeled the CCAs in a simulation environment in order to study their performance under certain workload situations. The RALE, Ramon Llull Environment, has been used [4] with its PADD notation, Parallelism Abstraction and Dimensional Design, a notation with explicit parallelism and synchronous message passing. The performance metric used is the Throughput or transaction completion rate. Table 2 gives the values of the key simulation parameters.

Table 2. Simulation Parameter Settings

Parameter	Setting	Parameter	Setting
NumClients and NumSites	8	DiskTime	1 ms
ThinkTime	0..1 sec	WaitCPU	0.03 ms
UpdateRatio	0.25	WaitCheckDD	10..1000 ms
TransactionClass	1, 2, 3, 4, 5	NetworkLatency	0.1, 0.5, or 1 ms

The **UpdateRatio** is computed by dividing the number of updates by the number of reads. The **Transaction Class** defines the composition of transactions used to send to the processors. With **TransactionClass** equal to 5, there are no possible conflicts between the transactions. With **TransactionClass** equal to 1, the probability of having conflicts between the transactions is very high, and the transactions use fully replicated information. With **TransactionClass** 2, the probability of conflicts is high, it is medium with **TransactionClass** equal to 3, and it is low with **TransactionClass** equal to 4.

The ThinkTime parameter is the time between the completion of a transaction and the submission of the next one by the Client. This parameter is varied over a range from 0 to 1 s, in order to influence the workload of the system.

The global deadlock detection interval WaitCheckDD is fixed at simulation time. The NetworkLatency is the response time of the network. We should also comment that in our simulations, we have considered a system with an I/O predominant cost parameter. The disk average access time DiskTime, considering buffering of 80%, is 1 ms, and the WaitCPU equal to 0,03 ms. In the remainder of this section, we show the results of our simulations.

5.1 Experiment 1: Effect of NetworkLatency

This experiment examines the impact of the NetworkLatency on all the algorithms. Figure 1 shows the system throughput with a NetworkLatency of 0,1 ms whereas figure 2 shows the throughput with a NetworkLatency of 1 ms.

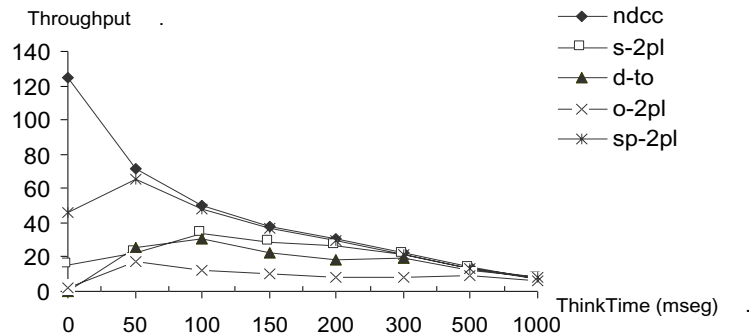


Fig. 1. Throughput with NetworkLatency of 0,1 ms and TransactionClass equal to 3

Since ThinkTime is used to modify the system load, going from right to left the load increases. At ThinkTime equal to zero, as experiment 1 shows, the system throughput is very low, and when the ThinkTime grows, the throughput also grows. But from ThinkTime = 50 ms or ThinkTime = 100 ms, the throughput goes down until ThinkTime = 1 s. This happens because the system load decreases, and all the CCA tend to the same throughput value.

As shown in Figure 2, with a NetworkLatency of 1 ms, sp-2pl also outperforms the rest of the CCA, giving the best throughput. But in this case, the differences between algorithms decrease as the NetworkLatency increases. The reason for this is that the amount of effort wasted on concurrency control becomes a smaller fraction of the total cost which includes network communications. As a result of this experiment, sp-2pl gives the best results on a system based on a fast

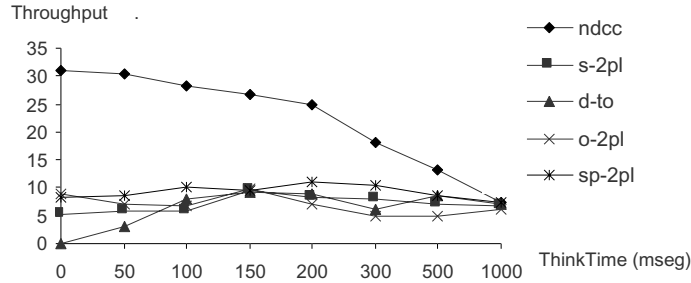


Fig. 2. Throughput with NetworkLatency of 1 ms and TransactionClass equal to 3

interconnection network or on a bus. But these results must be analyzed in detail with respect to the variation of the level of concurrency.

5.2 Experiment 2: Effect of TransactionClass

This experiment examines the impact on performance of an increase in system workload. Figure 4 shows the results obtained by repeating the simulations of experiment 1 with all the CCA, with a fixed NetworkLatency of 0,1 ms. As in experiment 1, sp-2pl outperforms all the algorithms.

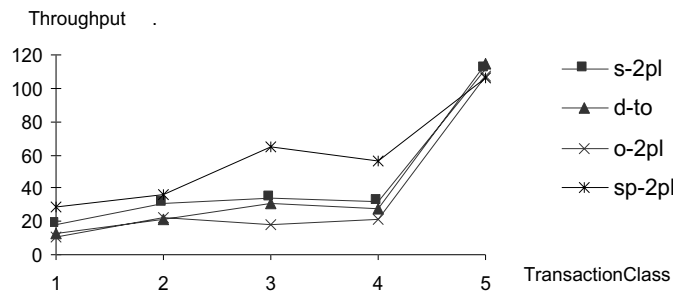


Fig. 3. Throughput. NetworkLatency of 0,1 ms

5.3 Experiment 3: Impact of WaitCheckDD

This experiment examines the impact of WaitCheckDD on throughput. Experiments 1 and 2 used a WaitCheckDD of 50 ms. If the WaitCheckDD is low, the system will detect before the possible deadlocks, but the network overhead

will be high. If the WaitCheckDD is high, the network overhead is low but the system will detect the deadlocks later.

Figure 4 shows the system throughput with a WaitCheckDD from 10 ms to 1 s. All the simulations of this experiment have been done with a Transaction-Class equal to 3 which corresponds to a medium workload. The sp-2pl-100 values correspond to a ThinkTime of 100 ms. In this situation, since the restart ratio is very low, the effect of WaitCheckDD on throughput is minimum. With low values of WaitCheckDD there is no network overhead, because we use a NetworkLatency of 0,1 ms, and with high values of WaitCheckDD, since there are no Global Deadlocks, the system throughput is constant. Looking deeper, if we study the same algorithm but with a ThinkTime equal to zero (sp-2pl-0), there are many more aborts due to deadlocks. So at high values of WaitCheckDD the throughput goes down, because the global deadlocks are not detected on time. In this scenario, when the WaitCheckDD is low, the throughput is also worse, because of the increase in the effort of detection and resolution of system aborts.

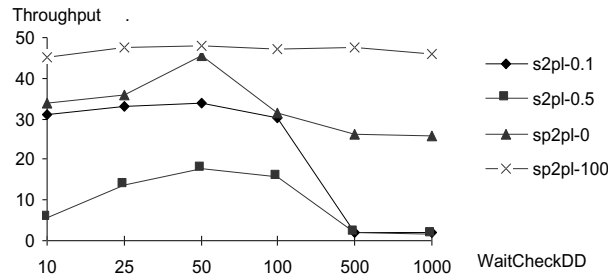


Fig. 4. Throughput with WaitCheckDD

Figure 4 also shows the effect of WaitCheckDD on s-2pl. The throughput with a NetworkLatency of 0,1 ms only goes down for high values of WaitCheckDD, as in sp-2pl-0. But with a value of NetworkLatency of 0.5 ms, the system throughput goes down for both low and high values of WaitCheckDD, as in sp-2pl.

6 Algorithm benefits

As we have seen in experiments 1, 2 and 3, sp-2pl outperforms all the other CCA. These improvements are based on the ECLM which allow some transactions to advance operations and to commit before than in s-2pl. The additional cost of sp-2pl is in the commit operation, in which a TM that coordinates the transaction is involved in a 2PC protocol. In this protocol, all the CCMs that have participated in the transaction have to validate all the Speculative operations executed during the transaction. This means that there is an increase in network cost at the end of the transaction, which does not exist in s-2pl.

As expected, Figure 1 indicates that sp-2pl is the best CCA over a fast network or a bus, but if we observe Figure 2, it is evident that the gap between the algorithms has decreased. To summarize, the sp-2pl algorithm advances the execution of operations that have a high probability of being compatible at the end of the transaction, avoiding possible end-of-transaction aborts and increasing the system throughput.

7 Concluding remarks and future work

In this paper we have studied the performance of a new concurrency control algorithm, sp-2pl. We have described the sp-2pl algorithm in detail and its performance has been studied with different system workloads. More details of our algorithm can be found in [5]. We found that sp-2pl outperforms s-2pl and the rest of CCA. However, when the NetworkLatency was high, the behaviour of sp-2pl tends to be similar to the rest of algorithms. Our techniques could be immediately applied in parallel and distributed databases. They may offer even more benefits by reducing the system contention in transactional environments.

We did not consider system failures in our simulations, so an open study is to extend the sp-2pl in order to be failure-tolerant. In addition to our conclusions about the sp-2pl and the rest of CCA, we think that an extension of our algorithm could be done using Wait-Depth Limited WDL techniques as in [8]. Finally we think that sp-2pl will be an ideal alternative in concurrency control algorithms.

References

1. D.Agrawal et al., The Performance of Protocols Based on Locks with Ordered Sharing, IEEE Trans. on Knowledge and Data Engineering, Vol.6, No.5, 1994.
2. P.A.Bernstein, D.W.Shipman, J.B.Rothnie, Concurrency Control in a System for Distributed Databases SDD-1, ACM Transactions on Database Systems, 1980.
3. P.A.Bernstein and N.Goodman, Timestamp-based algorithms for concurrency control in distributed database systems, 6th Int.Conf. on Very Large Data Bases, 1980.
4. M.Bertran et al., "RALE: An Environment for Simulation and Transformations" in *Real-time Systems; Modelling, Design and Applications*, World Scientific, 2001.
5. A.Climent et al., Alternative Strategies for Concurrency Control in Distributed Database Systems, Technical Report TR/2000-10, Engenharia La Salle, 2000.
6. M.Stonebraker, D.Skeen, A Formal Model of Crash Recovery in a Distributed System, IEEE Trans. on Software Engineering, Vol.9, No.3, pp.219-228, 1983.
7. A.Thomasian, Distributed Optimistic Concurrency Control Methods for High-Performance Transaction Processing, IEEE Trans. on Knowledge and Data Engineering, Vol.10, No.1, pp.173-188, 1998.
8. A.Thomasian, Concurrency Control: Methods, Performance, and Analysis, ACM Computing Surveys, Vol.20, No.1, pp.70-119, March 1998.
9. P.Valduriez and M.T.zsu, *Principles of Distributed Database Systems*, Prentice-Hall, 1999.
10. P.Valduriez, *Parallel Database Systems: Open Problems and New Issues*, Distributed and Parallel Databases, Vol.1, No.2, pp.137-266, April 1993.