Integrated Predicative Synchronization Implemented in a Federated DBMS*

Stefan Böttcher
IBM Deutschland GmbH
Scientific Center
Institute for Knowledge Based Systems
Postfach 80 08 80
D-7000 Stuttgart 80
West Germany
e-mail: BOETTCHE@DS0LILOG.bitnet

Abstract

An important implementation aspect of federated database systems is the implementation of the transaction synchronization component. Most approaches in database transaction synchronization are based on a single synchronization strategy, e.g. locking or validation. However, the adequacy of a transaction synchronization strategy often depends on the expected probability of conflicts. Therefore, it seems to be advantageous to adapt the synchronization strategy to the expected conflict probability. This paper describes an integrated scheduler based on predicative locking and predicative validation, which adapts its synchronization strategy to the expected conflict probability.

A comparison of predicative locking and predicative validation shows that predicative locking is superior to predicative validation if many conflicts occur, whereas predicative validation performs better if no conflicts occur. In order to combine the advantages of both synchronization strategies and to avoid their disadvantages, we develop an integration of both strategies. The integrated synchronization strategy uses an efficient heuristics in order to choose the appropriate synchronization strategy. An implementation of all three predicative synchronization strategies (locking, validation and their adaptive integration) within a single federated database system shows that the integrated synchronization is superior to both predicative locking and predicative validation.

*This work has been partially done at the University of Frankfurt and has been supported by the Deutsche Forschungsgemeinschaft under Grant-No SCHM350/3-1.

1 Introduction

A federated database system consists of a server containing the database and a couple of workstations which concurrently execute transactions of application programs. The transactions are synchronized by a scheduler the main part of which is located on the server and schedules the access to the database. We argue that the appropriate synchronization strategy for accesses to this database may be very much application dependent and therefore the server should integrate more than one synchronization strategy.

This paper presents an integrated predicative synchronization strategy based on both predicative locking and predicative validation. This integrated synchronization uses an efficient heuristics in order to choose an appropriate synchronization strategy. The heuristics is motivated by a comparison of predicative locking and predicative validation. It is further confirmed by performance tests for a given set of transactions. The performance tests compare predicative locking, predicative validation, and the presented integration of both using an implementation of these strategies within the DBPL database system [Böttcher, 1989]. Previous performance evaluations in the DBPL database system have shown that the time needed for predicative locking is less than 0.1 % of the transaction run time on the database server, if predicative locking uses an incomplete theorem prover [Böttcher *et al.*, 1986]. Therefore predicative synchronization is considered to be not too expensive.

Correctness of transaction scheduling and a high performance are considered to be the most important properties of a synchronization strategy for transactions accessing a relational database (e.g. [Papadimitriou, 1986]). Correctness of transaction scheduling and especially the phantom problem [Eswaran *et al.*, 1976], [Bernstein *et al.*, 1981], require to use predicative synchronization instead of physical synchronisation. On the other hand, performance comparisons have only been given for physical synchronization (e.g. [Franaszek and Robinson, 1985]) but not for predicative synchronization.

Performance results comparing predicative locking and predicative validation can not be expected to be the same as for a comparison of physical locking and physical validation, because physical locking uses a simple locking mechanism whereas predicative locking uses a fast theorem prover (e.g. [Böttcher *et al.*, 1986]) in order to check lock compatibility. Similarly, performance results comparing predicative locking and predicative validation with an integration of both may differ from a comparison of physical locking and physical validation with an integration of both of these strategies.

That is why this paper describes an integrated predicative synchronization strategy, and furthermore summarizes the result of a performance test for a given set of transactions, which compares an implementation of predicative locking, predicative validation and the presented integration of both.

The presented integration of locking and validation can not only be applied to queries and write operations on relational databases, but also to object-oriented databases using attribute inheritance [Böttcher, 1990a] and to the synchronization of integrity constraints and write operations [Böttcher, 1990b].

The paper is organized as follows. The next section summarizes the differences between predicative synchronization and physical synchronization. The third section discusses the advantages of both synchronization strategies and presents an integrated synchronization strategy. The fourth section characterizes the implementation of the integrated synchronization strategy and outlines the results of a performance comparison.

2 Predicative synchronization

The phantom problem is solved by both predicative locking and predicative validation, but not by physical locking or physical validation [Eswaran *et al.*, 1976], [Reimer, 1983].

2.1 The difference between predicative locking and physical locking

Detailed descriptions of predicate locking are given e.g. in [Eswaran *et al.*, 1976], [Bernstein *et al.*, 1981]. The difference of predicate locking compared to physical locking is that transactions lock predicatively described subrelations instead of tuples or physical relations. *Subrelations* are subsets of the relation schema of the corresponding relation. Two locks on subrelations of the same relation are compatible, if the subrelations locked do *not overlap*, i.e. if there does not exist any tuple in the corresponding relation schema which is contained in both subrelations. This overlapping of subrelations is checked by a fast theorem prover.

In order to get a fast theorem prover for predicative locking, we allow that it is incomplete, i.e. the theorem prover may decide, that it can not find out whether or not two subrelations overlap. In this case (and if the subrelations overlap), the scheduler delays one of the transactions. Only if the theorem prover finds out that both transactions can run in parallel, the scheduler allows them to run in parallel. As described in [Böttcher *et al.*, 1986] the theorem prover has a complexity of $O(n^3)$ and finds out all typical cases where both transactions can run in parallel.[1]

The time needed for predicative locking could be reduced to less than 0.1 % of the transaction run time, because the DBPL database uses an incomplete theorem prover. Therefore predicative synchronization is considered to be not too expensive.

2.2 The difference between predicative validation and physical validation

Predicative validation is described in [Reimer, 1983]. The division of transactions into phases, the assignment of transaction numbers, and the rule which transaction has to validate against which other transactions are equal to the parallel validation algorithm given for physical validation [Kung and Robinson, 1981]. The difference between predicative validation and physical validation lies in the evaluation of the validation condition.

As in physical validation, every transaction allocates a read set and a write for every relation it accesses. In

[1]The cases, where the theorem prover can not find out in a time $O(n^3)$ whether or not two subrelations overlap, are only of theoretical interest, but usually do not occur in practice.

predicative validation, the read set of a database relation contains the subrelations accessed by queries of the transaction[2], and the write set contains old and new values of tuples written by the transaction. The validation condition of predicative validation is the following: Two operations of two transactions are in conflict, if there is a subrelation contained in the read set of one transaction and there is a tuple in a write set of the other transaction so that the tuple is contained in the subrelation. This containment is checked by the query evaluation system.

Because of these differences between predicative validation and physical validation and between predicative locking and physical locking a comparison of predicative validation and predicative locking may lead to different results from a comparison of physical validation and physical locking.

3 Comparing and integrating predicative validation and predicative locking

On the basis of a discussion of the advantages and disadvantages of predicative validation and predicative locking, we present an integration of both strategies with the aim to combine the advantages and to avoid the disadvantages.

3.1 Comparing predicative locking with predicative validation

The following discussion of predicative locking and predicative validation extends a discussion in [Brägger and Reimer, 1983]. The disadvantages of predicative locking compared to predicative validation are:

- If predicative locking grants locks which are larger than necessary, then parallel transactions are blocked although they do not conflict. If on the other hand predicative locking requires locks as small as possible, then the time for checking lock compatibility may become very long.
- Every lock compatibility check has to be performed on a global data structure and has to be synchronized against lock compatibility checks of parallel transactions. This decreases parallelism.
- Incremental locking may lead to deadlocks. On the other hand, a deadlock-free locking policy reduces parallelism.

[2]The read set furthermore contains the subrelations characterizing the modified parts of the database relation.

- Two-phase locking requires to lock objects longer than they are accessed. This delays other transactions which could run in parallel under predicative validation.

The advantages of predicative locking over predicative validation are:

- If the transactions lock subrelations described by a simple selection condition, then it is faster to check whether these subrelations overlap or not than to use queries during the validation phase.
- Validating transactions are backed up and restarted, if serializability is endangered. With an increasing number of conflicts, this leads to a considerable amount of unsuccessful transactions. These unsuccessful transactions withdraw resources like CPU-time and disk accesses from successful transactions.

Performance evaluations using an implementation of both strategies in one single system, i.e. the DBPL database system developed at the University of Frankfurt [Böttcher, 1989], confirm especially what follows: Predicative locking is superior to predicative validation for histories with a high ratio of conflicts, whereas predicative validation performs better when very few conflicts occur (see also section 4.3). Therefore, it seems to be advantageous to integrate both synchronization strategies and to choose the appropriate synchronization strategy depending on the expected amounts of conflicts. This approach is described in the next subsection.

3.2 Integrated synchronization based on predicative validation and predicative locking

We first outline the idea of the integrated synchronization and afterwards give an example.

A key to the idea behind the presented integration is that the conflict probability of operations depends on the size of the subrelations accessed by the operations, i.e. usually tuple operations have a lower conflict probability than set-oriented operations accessing large parts of a relation.

A transaction may contain some operations with a lower conflict probability, i.e. tuple operations, and other operations with a higher conflict probability, e.g. set-oriented operations. Therefore, we do not require that this transaction uses the same synchronization strategy for each of its operations. Instead, we allow that some operations are synchronized by validation, whereas other operations of the same transaction are

	tuple operation	set-oriented operation
tuple operation	(predicative) validation	(predicative) locking
set-oriented operation	(predicative) locking	(predicative) locking

Figure 1: Selected Synchronization strategy depending on the operations.

synchronized by locking. This means that the synchronization strategy needs not to be chosen at the level of transactions, instead it can be chosen at the level of operations.

Furthermore, a given operation (e.g. a tuple operation) may have different conflict probabilities with different other operations, i.e. conflicts with other tuple operations are less probable than conflicts with set-oriented operations. Therefore, we allow that the given operation is synchronized with one strategy (say validation) against other tuple operations and with another strategy (say locking) against set-oriented operations.

The overall idea behind the integration is the following: The scheduler determines the adequate synchronization strategy for each pair of possibly conflicting operations of different transactions. If the conflict probability of this pair of operations is high, then the operations are synchronized by (predicative) locking, if on the other hand the conflict probability of this pair of operations is low, then the operations are synchronized by (predicative) validation.

Since the scheduler has to decide for every pair of operations whether their conflict probability is high or low, it needs a very fast heuristics for that decision. The heuristics which is implemented in the DBPL database system in order to select the synchronization strategy works as follows. The heuristics distinguishes two kinds of operations on database relations supported by the database programming language DBPL: tuple operations and set-oriented operations. Two tuple operations are only in conflict, if they operate on the same tuple. That is why the heuristics assumes, that the probability of conflicts between two tuple operations is low. On the other hand, a set-oriented operation conflicts with another operation, if the accessed subrelations of both operations overlap. That is why the heuristics assumes that the probability of conflicts between two operations is high, if at least one of them is a set-oriented operation.

The heuristics can be regarded as a voting mechanism: Each tuple operation votes for validation and each set-oriented operation votes for locking. Furthermore, the vote for locking overrides the vote for validation. Hence, if both operations are tuple operations, both operations vote for validation and they are validated against each other. If however one operation is a set-oriented operation, then this operation votes for locking and both operations are synchronized against each other by locking. The heuristics is summarized in the table given in figure1.

Now we give an example in order to illustrate the heuristics: Consider three transactions T1, T2, T3 accessing the same relation. T1 and T2 perform a tuple operation and T3 performs a set-oriented operation. Then the DBPL heuristics assumes that the conflict probability between the operations of T1 and T2 is low, while the conflict probability of the operation of T3 with the operation of T1 (and the operation of T2 respectively) is high. This means, the DBPL scheduler synchronizes the operations of T1 and T2 by validation, whereas the set-oriented operation of T3 is synchronized against the tuple operation of T1 (and the tuple operation of T2 respectively) by (predicative) locking.

In the integrated synchronization[3], one of both strategies (predicative locking or predicative validation) can be chosen for any single pair of operations independent from the synchronization strategies chosen for other pairs of operations. The correctness of this integrated synchronization is stated in the following theorem.

Theorem:

> If a scheduler uses the algorithm outlined in section 4.2 and synchronizes each pair of conflicting operations of different transactions either by (predicative) two-phase locking or by (predicative) validation, then every history produced by the scheduler is serializable.

The proof of this theorem is given in [Böttcher, 1989]. It is an extension of the serializability proof for two-phase locking, which can be found e.g. in [Bernstein *et al.*, 1987]. Since the proof does not depend on properties of predicative synchronization, the theorem holds for physical synchronization as well.

[3]The implementation of the scheduler based on integrated synchronization is described in the sections 4.1 and 4.2.

	p-read-lock	p-write-lock	e-read-lock	e-write-lock
p-read-lock	compatible	compatible	compatible	not compatible
p-write-lock	compatible	compatible	not compatible	not compatible
e-read-lock	compatible	not compatible	compatible	not compatible
e-write-lock	not compatible	not compatible	not compatible	not compatible

Figure 2: Compatibility of p-locks and e-locks.

4 Implementation and experimental results

Having outlined the idea behind integrated predicative synchronization, we now describe the implementation of the heuristics within the DBPL database system, sketch the transaction scheduling algorithm, and present a performance test, which demonstrates that at least for the tested transaction load the integrated scheduling is superior to both predicative locking and predicative validation.

4.1 Implementation of the heuristics

Since predicative validation is already described in [Reimer, 1983] and the fast theorem prover used for lock compatibility checks in the DBPL database system is described in [Böttcher *et al.*, 1986], here we restrict the implementation description to the heuristics selecting the synchronization strategy.

In order to implement the heuristics, the lock table of the DBPL database system distinguishes between two kinds of locks, exclusive locks (*e-locks*) and participation locks (*p-locks*). e-locks forbid other locks on overlapping parts of a relation, whereas p-locks forbid e-locks on overlapping parts of a relation, but allow for other p-locks on the same relation. Pairs of operations which are locked by p-locks are synchronized by predicative validation. The heuristics uses e-locks for set-oriented operations and p-locks for tuple operations. The lock compatibility is summarized in the table of figure 2 which also distinguishes read locks and write locks. This compatibility matrix guarantees that p-locks only have to be checked against e-locks but not against other p-locks. Hence, if two operations are synchronized by validation, i.e. they use only p-locks, then no lock compatibility check[4] is needed. Lock compatibility only has to be checked, if two operations have a high conflict probability and are synchronized by predicative locking, i.e. if at least one of them is a set-oriented operation and requires an e-lock.

[4]That is the most expensive operation of predicative locking.

In a similar way validation is restricted to be performed only for those pairs of operations, with a low conflict probability, i.e. tuple operations for which a p-lock was required. In order to implement this restricted validation, the integrated scheduler keeps in the read sets only used subrelations of the operations with a low conflict probability, and similarly it transfers only the old and new values of these operations into the write sets. Since the integrated synchronization does not collect the old and new values of set-oriented write operations in the write sets of the transactions (as predicative validation does), queries during the validation phase are applied to much smaller write sets than this is the case with predicative validation.

To summarize: Every pair of conflicting operations is synchronized by only one synchronization strategy, predicative locking or predicative validation. A fast heuristics selects the appropriate strategy depending on the expected conflict probability, i.e. depending on the kinds of both operations. If the conflict probability between them is high, then locking is applied. If on the other hand operations are synchronized by validation using queries, then these queries only have to be applied to small write sets.

4.2 The scheduling algorithm for integrated synchronization

In the following sketch of the transaction scheduling algorithm, `< A >` denotes that `A` is executed in a critical section. The algorithm is a combination of the parallel validation algorithm of [Kung and Robinson, 1981], the two-phase locking algorithm and the two-phase commit protocol [Bernstein *et al.*, 1987]:

```
< tBegin > ;
read phase ;
< start validation > ;
validation phase( successful validated );
prepare to commit( successful validated );
IF successful committed THEN write phase ;
END ;
< tEnd > ;
```

The procedures **prepare_to_commit** and **successful commited** implement the two-phase commit protocol [Bernstein *et al.*, 1987]. The procedures **tBegin**, **start validation** and **tEnd** are implemented as described in [Kung and Robinson, 1981]. During the read phase write operations are performed on local copies. Read locks are required immediately before the read operations, and write locks are required at the end of the read phase (before start_validation).[5] All locks are released at the end of transaction (before tEnd). In the validation phase only tuple operations (p-locked operations) are validated against other tuple operations (p-locked operations). Therefore, we expect that the validation time of integrated synchronization is rather short compared to ordinary validation.

4.3 A performance test within the DBPL system

Performance results comparing predicative locking and predicative validation can not be expected to be the same as for a comparison of physical locking and physical validation, because physical locking uses a simple lock compatibility checking mechanism whereas predicative locking uses a fast theorem prover (e.g. [Böttcher *et al.*, 1986]) in order to check lock compatibility.

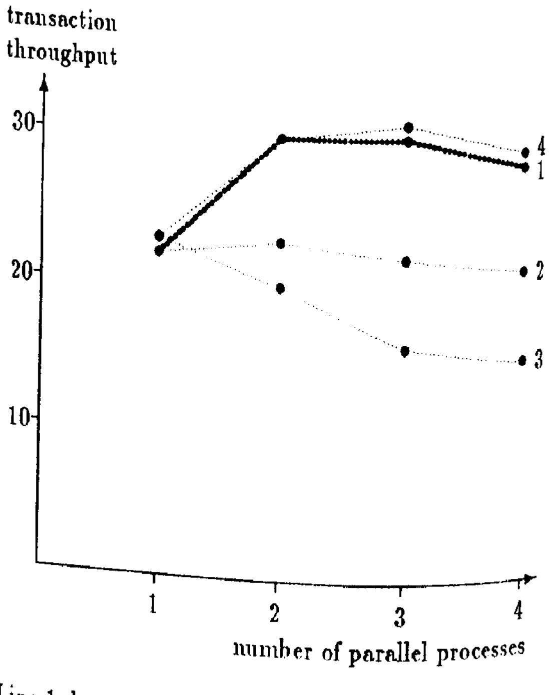
A performance comparison of predicative locking and predicative validation should consider the peculiarities of both synchronization strategies. E.g., predicative validation uses CPU-time and database accesses for aborted transactions, however predicative locking does not use these resources for waiting transactions. Furthermore, predicative locking uses a theorem prover for checking lock compatibility, while predicative validation uses query evaluation in the validation phase. Hence it seems to be very difficult to develop a model which not only considers the differences between both synchronization strategies but also correctly weights them. Modelling the costs for integrated synchronization may be even more complicated. Therefore, we preferred to implement all three strategies within one single database system[6] and to compare their performance. This was done on a MicroVAX II under VMS.

The performance test is as follows. We have a varying number of processes (from 1 to 4) infinitely submitting the same kind of transactions. Each transaction performs two write operations on single tuples and an integrity check containing a complex query. The query (written in DBPL [Schmidt and Matthes, 1990]) is as follows.

```
NOT SOME t1 IN R1 SOME t2 IN R2 SOME t3 IN R3
   ((t1.attr12 = t2.attr12) AND
   (t1.attr13 = t3.attr13) AND
   (t2.attr23 = t3.attr23)
```

Whenever locking is used, the subrelations locked for the write operations overlap with the subrelations locked for the query. For the predicative validation, the test distinguishes two cases, first that no conflict occurs at all, and second the more realistic case that some conflicts occur.[7]

We count the number of successful completed transactions within a fixed time interval (transaction troughput) and compare the integrated synchronization with predicative locking and predicative validation. The following diagram shows the transaction throughput depending on the number of parallel processes which submit transactions.



Line 1 denotes the integrated synchronization,
line 2 denotes predicative locking,
line 3 denotes predicative validation with conflicts,
line 4 denotes predicative validation without conflicts.

In this performance test, predicative validation wins if no conflicts occur at all, but looses if conflicts between the query and the write operations of parallel

[5] For the reason see the correctness proof in the appendix.

[6] We used the DBPL database system developed at the University of Frankfurt.

[7] This distinction does not apply to locking or integrated scheduling, since the subrelations locked for the query always overlap with the subrelations locked for the write operations.

transactions occur. Integrated synchronization is superior to both predicative locking, and it is also superior to predicative validation, if there occur some conflicts. This performance evaluation has been confirmed by other performance evaluations within the DBPL system which yield similar results [Kupijai, 1988].

5 Summary and Conclusion

We have compared predicative locking and predicative validation and have shown, that predicative locking is superior to predicative validation if many conflicts occur, whereas predicative validation performs better if no conflicts occur. In order to combine the advantages of both synchronization methods and to avoid their disadvantages, we have developed the integrated synchronization. Integrated synchronization uses a fast heuristics and schedules pairs of operations depending on their conflict probability: Pairs of operations with a low conflict probability, i.e. pairs of tuple operations, are validated against each other, whereas pairs of operations with a high conflict probability, i.e. pairs including at least one set-oriented operation, are synchronized by predicative locking.

Furthermore, we have implemented all three synchronization strategies, predicative locking, predicative validation and the integrated synchronization within the federated DBPL database system and compared their performance. The presented result shows that integrated synchronization is superior to both predicative locking and predicative validation. Altogether, integrated synchronization seems to be an important improvement to schedulers of federated database systems.

References

[Bernstein *et al.*, 1981] P.A. Bernstein, N. Goodman, and M.Y. Lai. *Laying Phantoms to Rest (By Understanding the Interactions Between Schedulers and Translators in a Database System).* Technical Report, Harvard University, Aiken Computation Laboratory, Cambridge, 1981.

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

[Böttcher, 1989] S. Böttcher. *Prädikative Selektion als Grundlage für Transaktionssynchronisation und Datenintegrität.* PhD thesis, FB Informatik, Univ. Frankfurt, 1989.

[Böttcher, 1990a] S. Böttcher. Attribute inheritance implemented on top of a relational database system. In M. Liu, editor, *Proc. 6th International Conference on Data Engineering*, Los Angeles, California, USA, 1990.

[Böttcher, 1990b] S. Böttcher. Improving the concurrency of integrity checks and write operations. In S. Abiteboul and P. Kannellakis, editors, *Proc. Third International Conference on Database Theory*, Paris, France, 1990. (to appear).

[Böttcher *et al.*, 1986] S. Böttcher, M. Jarke, and J.W. Schmidt. Adaptive predicate managers in database systems. In *Proceedings of the 12th International Conference on Very Large Data Bases*, Kyoto, Japan, 1986.

[Brägger and Reimer, 1983] R.P. Brägger and M. Reimer. *Predicative Scheduling: Integration of Locking and Optimistic Methods.* Report 53, ETH Zürich, 1983.

[Eswaran *et al.*, 1976] K.P. Eswaran, J.N. Gray, R.A. Lorie, and I.L. Traiger. The notions of consistency and predicate locks in a database system. *Communications of the ACM*, 19(11), 1976.

[Franaszek and Robinson, 1985] P. Franaszek and J.T. Robinson. Limitations of concurrency in transaction processing. *ACM ToDS*, 10(1), 1985.

[Kung and Robinson, 1981] H.T. Kung and J.T. Robinson. On optimistic methods for concurrency control. *ACM ToDS*, 26(2), 1981.

[Kupijai, 1988] N. Kupijai. *Kombination und Mischung von prädikativem Sperren und prädikativer Validierung.* Diploma thesis, University of Frankfurt, 1988.

[Papadimitriou, 1986] C.H. Papadimitriou. *The Theory of Database Concurrency Control.* Computer Science Press, Rockville, 1986.

[Reimer, 1983] M. Reimer. Solving the phantom problem by predicative optimistic concurrency control. In *Proceedings of the 9th International Conference on Very Large Data Bases*, Firence, Italy, 1983.

[Schmidt and Matthes, 1990] J.W. Schmidt and F. Matthes. *DBPL Language and System Manual.* In. Document, Univ. Hamburg, 1990.