# A FLEXIBLE MECHANISM FOR CONCURRENCY CONTROL IN DATABASE SYSTEMS

MARCOS R. S. BORGES
NCE 1287

Julho,1987

Universidade Federal do Rio de Janeiro
Núcleo de Computação Eletrônica
Caixa Postal 2324
20001 - Rio de Janeiro, RJ
BRASIL

Este trabalho foi apresentado no XIV SEMISH (Seminário Integrado de Software e Hardware) durante o VII Congresso da Sociedade Brasileira de Computação realizado em Julho de 1987 em Salvador.

NCE UFRJ UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
NÚCLEO DE COMPUTAÇÃO ELETRÔNICA

## RESUMO

Uma das áreas na qual um grande esforço de pesquisa tem sido dediçado nos últimos anos refere-se ao problema do controle de concorrência em bancos de dados. Apesar do grande número de novos algoritmos, de nenhum deles até agora pode-se dizer.que funciona melhor para todos os tipos de situações. Este trabalho examina este problema e sugere um enfoque mais flexível para o projeto de mecanismos de controle de concorrência em sistemas de bancos de dados. Com a flexibilidade proposta tenta-se melhorar o desempenho do sistema em situações difíceis de serem tratadas eficientemente pela maioria dos mecanismos.



## ABSTRACT

One of the areas to which a great amount of research has been devoted in recent years is concerned with the concurrency control problem in a multi-user database system. Despite the great number of new algorithms, none of them so far could actually claim to work best in all types of applications. This article addresses this issue and suggests a more flexible approach to the design of concurrency control mechanisms in database systems. This proposed flexibility attempts to improve the performance of the system in a number of situations where the strictness of most mechanisms makes it difficult to achieve.

# I-INTRODUCTION

The purpose of the concurrency control activity in a database system is to ensure that the execution of one transaction will not interfere with the correct execution of another transaction running in parallel. At the same time, this control should be accomplished with the minimum interference to the normal course of transaction execution. This makes the concurrency mechanism a very important part of the database system [15].

The aim of this paper is to advance towards the design of a more flexible mechanism for concurrency control in database systems. The proposed flexibility attempts to produce a mechanism that should be able to perform better than the 'rigid' algorithms in a great number of situations.

The approach followed to produce a flexible mechanism is twofold. On one hand, it should define how the characteristics of the application that suggest a different behaviour on the part of the locking algorithm can be passed to the mechanism. It is assumed that the application designer will play an important role in identifying the characteristics and in supplying this information to the mechanism in the form defined.

On the other hand, the alternative protocols developed to deal with these special situations have to be made compatible with a standard concurrency control protocol which takes care of the situations in which a standard protocol is required. This approach means bringing less complexity to an already complex component of database systems.

Following the current consensus that places the locking approach as the one which produces the best overall performance [1, 2, 10], we decided to build the flexible mechanism starting from a locking algorithm. The study, which is presented in section III, concentrates on the weakness of the locking approach, especially on its pessimism for certain application types. The aim is to provide alternative routes in the locking mechanism either for increasing parallelism or reducing overhead in the system.

## II - PRINCIPLES OF THE CONCURRENCY CONTROL PROBLEM

### II.1 - The problem of parallel execution of transactions

When parallel processing is allowed in the system the actions of different transactions will interleave and this may result in problems in the database. The examples shown in Figures II.1 and II.2 illustrate the two common types of

problem arising out of the uncontrolled interleaving of the actions of two transactions [8].

Figure II.1 - Case 1: An example of the lost update problem

initial state: x=47

| order | T1 | result | T2 | result |
|---|---|---|---|---|
| 1 | read x into A | (A=47) | | |
| 2 | A = A + 2 | (A=49) | | |
| 3 | | | read x into B | (B=47) |
| 4 | write x from A | (x=49) | | |
| 5 | | | B = B + 3 | (B=50) |
| 6 | | | write x from B | (x=50) |

final state: x=50 (should be 52);
Transaction T1 was not reflected in the database.

Figure II.2 - Case 2: An example of an inconsistent updating

initial state: x=10; y=15; z=25; constraint: x + y = z

| order | T3 | result | T4 | result |
|---|---|---|---|---|
| 1 | read x into A | (A=10) | | |
| 2 | read z into B | (B=25) | | |
| 3 | A = A + 2 | (A=12) | | |
| 4 | B = B + A - x | (B=27) | | |
| 5 | write x from A | (x=12) | | |
| 6 | | | read x into C | (C=12) |
| 7 | | | read z into D | (D=25) |
| 8 | write z from B | (z=27) | | |
| 9 | | | C = C * 2 | (C=24) |
| 10 | | | D = D + C - x | (D=37) |
| 11 | | | write x from C | (x=24) |
| 12 | | | write z from D | (z=37) |

final state: x=24; y=15; z=37; (should be: x=24; y=15; z=39)
Transaction T4 read and stored inconsistent data.

In the first example, the problem is known as the lost update problem. One of the transactions did not have its results reflected in the database. In the second

example, although the transactions would maintain the integrity when processed separately, if the actions interleave in the way shown, an inconsistent state will result. The two situations are examples of illegal histories.

## II.2 - The correctness issue

Given an algorithm to deal with concurrent transactions it is necessary to verify whether the behaviour of the algorithm is correct, for example, if the algorithm avoids, in all circumstances, the situations described in the previous section. This definition gave rise to serializability theory [3, 20], which is a collection of mathematical rules that tell whether a concurrency control algorithm works correctly [5]. This subject, however, is out of the scope of this paper.

## II.3 - Approaches to the concurrency control problem

The approaches differ in many aspects, but mainly over the question of at which stage of the transaction execution the conflicts are detected and resolved. There are a number of variations originating from each basic algorithm and from some other algorithms which cannot be classified in any of the main approaches. The motivation for the variations, however, is on the grounds of efficiency.

### II.3.1 - The Locking Approach

In the locking approach every action performed on an object must be preceded by a lock on this object. The lock operation will prevent any other action from operating in this object for the duration of the lock. The locks are released by the transaction when the actions have been performed on the object. The transactions that follow the protocol: lock ---> action ----> unlock, are said to be "well formed". If transactions are "well-formed and the locking protocol is two- phased [12], the mechanism works correctly and is called two- phase locking mechanism.

### II.3.2 - The Optimistic Approach

The rules imposed on the transactions in the two-phase locking approach may be considered too strict because of the the inhibition of parallelism among transactions and the consequent costs involved [18]. The inability of the locking protocol to relax these rules when there is a low rate of conflict was the starting point for the class of algorithms named "optimistic" or "certifiers" [18]. The denomination "optimistic" results from the assumption that conflicts between transactions are rare and therefore, should not be prevented, but remedied when they occur. Instead of delaying a transaction because of the locking procedure, the mechanism allows it to run freely and only

checks for conflicts immediately before it is ready to commit.

### II.3.3 - The Timestamp Ordering Approach

The third category of concurrency mechanisms has a different attitude with regard to the order of transaction execution. In the timestamp ordering approach, the mechanism enforces a pre-fixed order which is determined when the transaction is submitted. The timestamp information is attached to the transaction for this purpose [4, 11, 21].

Conflicts can be prevented simply by imposing the order established by the timestamp information on all accesses to the database. This means that if all accesses are performed in the same order, which has been determined by the transactions timestamp, the cyclic situation will be avoided in the resulting serialization graph. The proof of correctness of the basic algorithm of this approach appears in [5].

## II.4 - General Remarks

There are a considerable number of algorithms for concurrency control in database systems. In the years to come several others are expected to appear. The reason for such proliferation of algorithms comes from the impossibility of defining appropriate requirements for the

design of the mechanisms. At the present stage, the correctness requirement covers a very restricted range of situations, and the performance requirement is naturally vague because of the number and the complexity of links between the concurrency control and the remaining components of the system.

## III - THE USE OF SEMANTIC INFORMATION IN THE DESIGN OF CONCURRENCY CONTROL MECHANISMS

The performance of the concurrency control mechanism is very sensitive to the type of transaction load submitted to the database system. Based on this hypothesis, most of the research on algorithm construction has been oriented towards developing an algorithm which performs well in most common types of application. The results of comparative analyses, however, have suggested that this seems to be an unattainable goal [9, 10].

An approach, which has been the subject of consideration for some time, is the introduction into the algorithm of some sensitivity to the semantics of the application. Several mechanisms which employ some kind of semantic knowledge have been proposed [4, 7, 14]. The development of most new algorithms, however, are still set on the idea that the mechanism should be completely transparent to the user at all levels.

Our view is that the concurrency control mechanism should be projected in such a way as to be sensitive to the characteristics of the application by means of accepting parameters supplied by the user. These parameters would inform the mechanism about the semantics of the application in order to guide it to the most appropriate protocol. The objective is to construct a flexible mechanism which should be able to perform well in a greater number of database applications.

In this section three types of semantic knowledge will be reviewed. For each of these types, we analyse the way information can be passed to the concurrency mechanism in order to make it sensitive to certain characteristics of the application.

### III.1 - The locking by-pass strategy

When actions of one transaction overlap with actions of an update transaction in the same or a related object, inconsistent data may result. In the case of Read-only transactions, however, some transactions may not require strictly consistent data; for example, transactions for the purpose of collecting statistics. For these queries there may be no need to prevent inconsistencies. Their Read actions may overlap with any other actions of an update transaction, thus allowing a higher level of parallelism.

A possible way to make the concurrency mechanism consider this aspect is by implementing alternative routes which will be followed according to the consistency level assigned to each transaction. This idea was first introduced in [17] and used in the design of System R [6].

The strategy proposed here consists of by-passing the locking procedures in the synchronization process. Because of the possibility of inconsistent data, this strategy can only be employed in Read-only transactions which are prepared to accept inconsistent data. The strategy is particularly advantageous to those transactions which access frequently updated objects [13].

An additional variable (consistency code) is included in the transaction descriptions. At the time of the transaction execution the user assigns to this variable one of the two possible values. The values specify whether the user is prepared or not to accept inconsistent data as a result of the transaction execution. The locking algorithm will be modified as follows:

At the beginning of the transaction the Transaction Manager accesses the consistency code assigned to the transaction. If this code is set to Level 1 of consistency, then all lock requests associated with this transaction will be skipped. No changes are necessary on the other

transactions or in the Scheduler. An additional test is required in order to prevent Level 1 of consistency being assigned to Update transactions.

The benefits of this strategy are potentially great. Firstly, there is the elimination of all locking overhead for transactions following Level 1 of consistency. Secondly, a higher level of parallelism can be achieved because overlapping between this type of transactions and Update transactions will be allowed. While in the normal locking protocol one of the transactions would be blocked because of the existence of incompatible locks in the same object, this will not happen in the by-pass strategy in which the shared lock is not requested.

Depending on the characteristics of the application the chances of retrieving inconsistent data may be greatly diminished. Inconsistent data will be generated only if an Update transaction has overlapped its access in common objects. This overlapping will not occur, for example, to transactions consisting of a single object access.

## II.2 - The suspended locking strategy

Database systems are often characterized by a set of objects that are commonly referenced by the majority of query and update applications. On the other hand, there are

some objects that work as static information in the system. These objects are expected to be updated only rarely. It is also a common phenomenon that certain categories of transactions run during a specific period of the day.

These two observations can be very useful in increasing the performance of a concurrency mechanism [8]. Taking these factors into account will not produce more direct parallelism, but it may reduce overheads and therefore a better performance may be achieved. In a rarely updated object, a normal locking mechanism would still issue locks on these objects because of a future update request. However, if an object is known to be rarely updated, then an optimistic protocol may be adopted.

In the suspending locking strategy, for selected objects, the locking requests are substituted by a post-validation schema based on the optimistic approach. The objects are selected by the application designer and not automatically by the mechanism. The designer should identify on behalf of the mechanism the objects in which a small proportion of update operations are expected. This information, together with other auxiliary information generated by the mechanism, will be stored in a table called Status Table.

Besides the relation identification, each entry in the table has three other variables. The first describes the present access status of the relation. The second describes the timestamp version of the data stored in the relation. The timestamp is simply an integer variable used to register the last occurrence of an update transaction processed in some object of the relation. Finally, there is a variable describing the number of active update transactions which have been granted an exclusive lock on some object of the relation.

The access status is set to one of the three possible values: regular locking (RL), suspended locking (SL), and temporary locking (TL). The other two variables are only accessed when the access status is set to 'SL'. When an action is due to start, the Transaction Manager accesses the Status Table in order to check the status of the relation accessed by the action. One of the following routes is possible:

If the access status variable of a relation is marked with a regular locking code ('RL'), then the action (either it is an Update or a Read-only action), is routed through the normal locking protocol, . Obviously, if all relations are set to 'RL' then the algorithm works in a similar way to the basic locking mechanism.

If the access status variable is set to 'SL', then this means that a low rate of conflicts is expected and the optimistic protocol should be followed for the synchronization of this action. For Read-only actions the locking mechanism is skipped. Before executing, however, the value of the timestamp version variable is copied into the action working space to be utilized later during the validation phase.

When an update action is requested on a relation marked with the 'SL' status, the optimism of the strategy should be temporarily suspended. The access status of the relation updated should then be set to the 'TL' value. This will prevent any future Read actions from following the optimistic route while an update action is in progress. In order to control the number of updates active on this relation, the value of the active update variable is increased by one for each started Update action and reset at the end of the transaction.

This strategy is a compromise between the pure locking and the optimistic approaches. The idea is to combine the two strategies into a single algorithm. The objective, as in the optimistic approach, is to achieve a reduction of the synchronization costs by suspending the locking action for some of the accesses [93. On the other hand, this strategy, unlike the optimistic algorithm, does not apply generally to

all accesses because it is recognized that, within the same application, normal locking is preferable in a number of situations.

### III.3 - Hierarchical locking

In the locking approach, conflicts are prevented by issuing a lock on the object accessed by the transaction. However, the size of the 'lockable' units is a matter of strategy for the concurrency mechanism. While a fine granularity of locking such as pages and records enables more potential parallelism, it also produces higher overheads than a coarse granularity locking [22].

The hierarchical locking approach was proposed in [17] describing the mechanism which was later used in System R [6]. In this approach a hierarchy of granules is defined. The transactions may request a lock on any of the levels of the hierarchy but all the higher levels must be acquired on an 'intention mode'.

The justification for this hierarchy is the recognition that the solution of the problem of optimal granularity is application dependent [23], and therefore a more flexible protocol is expected to perform better in a larger number of systems.

For the usage of this feature it is proposed that the assignment of granularity hierarchy (in our case the locking granularity hierarchy) should not be strictly defined. Instead, the application designer should have the means for altering the hierarchy in a way that suits most applications. In other words, this approach will allow the hierarchy to be specified according to the system implemented in order to achieve the best performance.

Instead of having a rigid table of hierarchies such as suggested by Gray et al [16], the database system will follow the hierarchy specified by the application designer. This will permit the strategy to vary from a non-hierarchy granularity to a detailed hierarchy of objects [19]. The variable hierarchy will be particularly useful in non-standard database applications where the types of objects differ from the usual database types (relation and pages).

## IV - CONCLUSIONS

We advocated in this paper the use of semantic knowledge of the application in the design of the concurrency mechanisms in order to provide the flexibility that would permit the user to tune the mechanism to suit the requirements of the application. This was based on the belief that only by introducing some sort of sensitivity to

the application can a mechanism be expected to work at its best in a wider range of situations.

In the selection of which type of semantic information to include in the mechanism, two major criteria prevailed: the generality of its use, and the relative simplicity of its implementation in an already complex mechanism. The association of these two qualities with the features of the locking mechanism is expected to produce a highly efficient mechanism for concurrency control in databases.

As demonstrated by the results produced in [9], the major benefit of such approach is a very significant improvement in the system's performance resulting from the combination of greater parallelism with smaller overhead in the concurrency control activity. In some cases, an increase of over 50% in the response time was achieved with the usage of th flexible approach. This was achieved with simple but very effective changes in the basic locking protocol directed at resolving its major drawbacks, such as its permanent pessimism regarding transactions overlap.

## V - REFERENCES


[ 1] Agrawal, R., Carey, M.J. and Livny, M. Models for studying concurrency control performance: Alternative & implications. In Proc. ACM SIGMOD International Conference on Management of Data, Austin, Texas, May 1985, pp. 108-121.

[ 2] Augustin, R., Pradel, U. and Scholten, H.A. Modelling database concurrency control algorithms using a general purpose performance evaluation tool. Technical Report Nr. 42, University of Hagen, Hagen, Feb. 1984, 26 pp.

[ 3] Bernstein, P.A., Shipman, D.W. and Wong, W.S. Formal aspects of serializability in database concurrency control. IEEE Transactions on Software Engineering 5, 3(May 1979), 203-216.

[ 4] Bernstein, P.A., Shipman, D.W. and Rothnie, J.B. Concurrency control in a system for distributed databases (SDD-1). ACM Transactions on Database Systems 5, 1(March 1980), 18-51.

[ 5] Bernstein, P.A. & Goodman, N. A sophisticate's introduction to distributed database concurrency control. In Proc. International Conference on Very Large Data Bases, Mexico, 1982, 62-76.

[ 6] Blasgen, M.W. et al. System R: An architectural overview. IBM Systems Journal 20, 1(1981), 41-62.

[ 7] Boral, H. & Gold, I. Towards a self-adapting centralized concurrency control algorithm. In Proc. ACM SIGMOD International Conference on Management of Data, Boston, MA., June 1984, pp. 18-32.

[ 8] Borges, M.R.S. Towards a flexible mechanism for concurrency control in database systems. In Proc. 4th British National Conference on Databases (BNCOD4), A.F. Grundy (ed.), Cambridge University Press, Cambridge, 1985, pp. 39-60.

[ 9] Borges, M.R.S. A flexible mechanism for concurrency control in database systems. Ph.D. Thesis, University of East Anglia, 1986, Norwich, Inglaterra.

[10] Carey, M.J. & Stonebraker, M.R. The performance of concurrency control algorithms for database management systems. In Proc. International Conference on Very Large Data Bases, Singapore, Aug. 1984, pp. 107-118.

[11] Casanova, M.A. The concurrency control problem for database systems. Lecture Notes in Computer Science Vol. 116, G. Goos & J. Hartmanis (eds.), Springer-Verlag, Berlin, 1981.

[12] Eswaran, K.P. et al. The notions of consistency and predicate locks in a database system. Communications of the ACM 19, 11(Nov. 1979), pp. 624-633.

[13] Garcia-Molina, H. & Wiederhold, G. Read-only transactions in a distributed database. ACM Transactions on Database Systems 7, 2(June 1982), 209-234.

[14] Garcia-Molina, H. Using semantic knowledge for transaction processing in a distributed database. ACM Transactions on Database Systems 8, 2(06/83), 186-213.

[15] Gardarin, G. & Melkanoff, M. Concurrency control principles in distributed and centralized databases. Rapports de Recherche 113, INRIA, Jan. 1982, 91 pp.

[16] Gray, J.N. et al. Granularity of locks and degrees of consistency in a shared data base. Research Report RJ1654, IBM Research Laboratory, San Jose, September 1975, 29 pp.

[17] Gray, J.N. Notes on database operating systems. In Operating Systems: An Advanced Course, Lecture Notes in Computer Science Vol. 60, R. Bayer et al. (eds.), Springer-Verlag, New York, 1978, pp. 393-481.

[18] Kung, H.T. & Robinson, J.T. On optimistic methods for concurrency control. ACM Transactions on Database Systems 6, 2(June 1981), 213-226.

[19] Page, T.W., Weinstein, M.J. and Popek, G.J. Genesis: A distributed database operating system. In Proc. ACM SIGMOD International Conference on Management of Data, Austin, Texas, May 1985, pp. 374-387.

[20] Papadimitriou, C.H. The serializability of concurrent database updates. Journal of ACM 26, 4 (Oct. 1979) 631-653.

[21] Reed, D.P. Naming and synchronization in a decentralized computer system. Ph.D. Thesis, Dept. of Electrical Engineering, MIT, Cambridge, Mass., September, 1978.

[22] Ries, D.R. & Stonebraker, M.R. Effects of locking granularity in a database management system. ACM Transactions on Database Systems 2, 3(September 1977), 233-246.

[23] Ries, D.R. & Stonebraker, M.R. Locking granularity revisited. ACM Transactions on Database Systems 4, 2(June 1979), 210-227.