Locking with Prevention of Cyclic and Infinite Restarting in Distributed Database Systems

Wojciech Cellary and Tadeusz Morzy

Institute of Control Engineering
Technical University of Poznan
Poznan, POLAND

ABSTRACT

A new solution to the cyclic restarting and infinite restarting problems for locking schemes in Distributed Database Systems (DDBSs) is presented. The solution proposed is based on the data marking mechanism, which ensures the completion of each transaction in the system. The solution is fully distributed. It only requires information locally accessible on each site of the DDBS, and it intervenes into transaction processing only in the case of real danger of cyclic and/or infinite restarting. Simulation has shown that this solution significantly reduces the number of transaction restarts in DDBSs using locking schemes, and thus considerably improves DDBS performance.

I. Introduction

Two general goals of concurrency control in transaction-based Distributed Database Systems (DDBSs) are: (I) to preserve database consistency, and (2) to guarantee the completion of each transaction submitted to the system in finite time. Let us note that both these goals concern consistency in the wide sense. The first one concerns the consistency of data stored in the database. The second one concerns the consistency of the database with respect to the external world which it reflects. To clarify this problem, let us consider the situation when a transaction updating a bank account can never be completed (for some reasons which will be explained later). The data stored in the database are consistent among themselves but the database is not consistent with respect to the external world since money was taken from or put into the account, but the account was not updated.

The commonly accepted solution to the concurrency control in distributed databases is the use of locking schemes [I, 6, 7, I3, 2I, 32]. It has been proved that among locking schemes, the Two-Phase Locking (2PL) scheme is the best one for general distributed database systems, admitting data dependent transactions[1], which can modify any data item they access [I5, 32]. The 2PL scheme preserves database consistency, but does not guarantee the completion of each transaction in finite time, and it should thus be supplemented by some special algorithms that do this.

To guarantee the completion of each transaction in finite time, we must protect each transaction from four kinds of phenomena which can prevent its completion, namely: deadlock, permanent blocking, cyclic restarting and infinite restarting[2].

Deadlock occurs when two or more transactions wait forever for each other.

Permanent blocking occurs when a transaction waits forever for data locking because of a steady stream of other transactions whose lock requests are always accepted before its lock request.

Cyclic restarting occurs when two

---

[1] By a data dependent transaction, we mean one for which, in general, not all of the set of data items accessed is known at the beginning of the transaction processing because it is determined during transaction processing based upon data accessed earlier.

[2] The last two phenomena are also known under the following names: dynamic deadlock, livelock, permanent restarting, cyclic restart, restarting forever.

or more transactions always cause the abortion of each other.

Infinite restarting occurs when a transaction requesting a data lock is always aborted because of a steady stream of other transactions whose lock requests are always accepted before its lock request.

It is easy to notice some symmetries between these phenomena. First, deadlock and cyclic restarting concern a set of transactions whose lock requests are contradictory in some sense while permanent blocking and infinite restarting concern transactions which are permanently postponed because of "bad" characteristics, in comparison with those of other transactions continuously arriving.

Second, deadlock and permanent blocking can occur when unlimited waiting is allowed in the case of lock incompatibility, while cyclic and infinite restarting can occur when unlimited number of abortions are allowed.

In the literature, the most attention was paid to the deadlock problem. There are two general approaches to this problem in DDBS, namely: explicit detection and recovery, and prevention.

There have been several deadlock detection protocols suggested, both centralized and distributed, which differ from one another in the algorithms used to locate directed cycles in the so called demand graph representing the global state of all transactions in progress in the system [I, 4, I0, II,I3, I7, 22, 30]. However, due to the huge number of system resources (data lock units), data replication and the inherent communication delays in distributed computer systems, it is quite difficult to construct and maintain the consistent demand graph for the whole distributed transaction system. Indeed, it was shown in [9, I2], that most of the distributed deadlock detection protocols proposed to date are incorrect. Even assuming the improvement of these protocols, their very poor performance renders them impractical for DDBSs.

Prevention is the alternative approach to the deadlock problem. The simplest deadlock prevention strategy consists of the abortion of a transaction requesting a data lock in any case of its incompatibility with the locks already granted. The aborted transaction is rolled back (i.e., all its locks are preempted and released) before it is restarted. A somewhat more sophisticated deadlock prevention strategy allows the transaction to wait a prespecified time period (called timeout) before being aborted [I, 2, 7, 8, I3, 3I]. Let us note that these strategies also prevent permanent blocking.

Another prevention strategy, used in such distributed systems as XDFS and EWFS is based on time-limited breakable locks [I8, 3I]. A transaction that has some of its locks "broken", will be aborted if it tries to commit broken locks, which of course prevents potential deadlock.

Summarizing, for the solution of the deadlock problem in the DDBS, a version of the prevention strategy using abortion is usually adopted, because of its simplicity in comparison with the deadlock detection, the control using only the information locally available at the DDBSs' sites, and the simultaneous prevention of permanent blocking.

However, as was mentioned above, the use of the abortion as the principal mechanism of deadlock and permanent blocking prevention provokes cyclic and/or infinite restarting, which are totally unacceptable as well [I, 4, 9, II, 20, 26, 27]. Thus, it is necessary to introduce some strategies for the prevention of these phenomena to the DDBS concurrency control.

There are some known solutions to the problems of cyclic and infinite restarting in DDBSs. The most important group of methods is based on priority schemes.

The best known priority schemes are Wait-Die (WD) and Wound-Wait (WW) both proposed by Rosenkrantz et al. [I, I3, 27], in which both waiting and abortion are used. These schemes are free from deadlock and cyclic restarting. However, they are not free from permanent blocking and infinite restarting[1].

Another solution is used in the XDFS system [I8], where the additional intention-write lock mode is introduced to the set of basic lock modes to reduce the likelihood of a cyclic and infinite restarting. Such a solution considerably reduces the probability of cyclic restarting, but does not eliminate it. Moreover, this solution does not eliminate infinite restarting.

---

[1] This results from the fact that, in general, in locking algorithms the requirements of the strict concurrency control assumed in the original proposition of WD and WW (27) are not preserved. Additional mechanisms are needed to restrict the concurrent reading of the data items and thus avoid the above problems. However, the performance of the locking algorithms then becomes very low [I9, 20].

A simple, but rather radical solution to the infinite restarting problem consists of cutting off the stream of new transactions initiated in the system. In other words, after the detection of a repeatedly restarted transaction, the requests of new transactions arriving to the system are suspended until after it reaches its commit point [I6, 26]. This is equivalent to write-locking the entire database, and thus, this method can be used only if infinite restarting occurs very rarely.

Summarizing, there has, until now, been no method for preventing transactions from all of: deadlock, permanent blocking, cyclic and infinite restarting, and which does not considerably reduce the level of concurrency. The most complete proposal up to date are those using priority schemes. However, the main disadvantage of these methods is that they always intervene in transaction processing, whether there is a real danger of unacceptable phenomena or not, thus reducing the level of concurrency and DDBS performance.

In this paper, we propose a new, simple and efficient cure for all the phenomena listed above, which intervenes only in the case when there is a real danger of them, thus keeping the level of concurrency in the DDBS high. Moreover, we show, that the application of this method can improve the total DDBS performance. The method is fully distributed and it only needs information locally available at DDBS sites and does not need any extra inter-site communication.

We use abortion after the prespecified time-out, in order to eliminate indefinite waiting and thus prevent the system from deadlock and permanent blocking, and we propose data item marking as an approach for preventing of transactions from cyclic and infinite restarting. To this end, the restart indicator is attached, to each transaction initiated in the system. This defines the admissible number of transaction restarts, above which a transaction is treated as restarting infinitely. When such a transaction is detected, the system, according to our method, reserves all data items needed by it in such a way that it will be completed with minimum system performance cost. The mean response time to a transaction and the mean number of transaction restarts were used as the system performance measures.

The paper is organized as follows. Section 2 contains the basic definitions of the DDBS model adopted. In Section 3, the proposed method for the prevention of cyclic and infinite restarting is described. In Section 4, the results of the simulation experiments are discussed. The performance of the proposed method is compared with classical 2PL. This section also contains the discussion of the choice of the appropriate value of restart indicator, which considerably influences the performance of the presented method. Section 5 contains conclusions.

2. A model of distributed database system

2.I. Basic definitions

We consider a distributed system as a collection of sites interconnected by a communication network and controlled by a Distributed Database Management System (DDBMS). A Distributed Database (DDB), formally, is defined as a triple DDB=(ST, L, D), where $ST=(ST_1,\ldots,ST_N)$ is a collection of sites, L is a set of logical data items called the logical database, and D is a set of physical data items called the physical database.

We assume, according to the above definition, a two-level DDB model. At the higher level of abstraction (i.e., from the user point of view), DDB is represented by the set of logical data items denoted $(X_1,\ldots,X_n)$. In practice, the logical data items may be relations, files, records, etc. Each logical data item may be represented by a set of one or more (replicated) physical copies stored at different sites of the DDBS. The physical copies of logical data item X_i are denoted $x_{i1}, x_{i2},\ldots, x_{in_i}$. Each physical data item may be represented by one or more physical units of access (e.g., pages of virtual memory). For simplicity, in this paper, we will assume the so called page-level access, i.e., the page is assumed to be a unit of locking. In practice, this assumption is usually met [3I].

The set of all physical data items $\{x_{ij}: 1\leq i\leq n, 1\leq j\leq m_i\}$, (i.e., physical database D) represents the DDB at the lower level of abstraction. Henceforth, by "data item", we will mean physical data item. Physical database D is the application-oriented implementation of L, for a given DDBS hardware configuration. Each site of DDBS accommodates the subset of the physical database D called the Local Database (LD).

Data items are operated on by transactions consisting of read and write operations and local computations. The "transaction" means a physical transaction here, not a logical one (i.e., a transaction after compilation and optimization).

In this paper, we assume the general multi-step transaction model [20, 24], in which transactions may be data-dependent. Moreover, a transaction is not necessarily a sequence of operations, but rather consists of (sequential and/or concurrent) subtransactions executed concurrently at different sites of the DDBS. A set of transactions residing in the DDBS is denoted $\tau = (T_1, \ldots, T_m)$.

Each site of the DDBS may contain two software modules: a Transaction Manager (TM), which initiates and supervises transaction processing, and a Data Manager (DM), which manages and controls access to a local database at a given site. Each LD is controlled by a single DM, which accepts subtransactions from TMs and is responsible for the concurrency control and recovery in its local database.

2.2. Transaction processing

Let us now briefly describe the general transaction processing scheme in accordance with 2PL in order to point out the possibility of cyclic and infinite restarting.

Our 2PL realization is similar to the one presented by Moss [2I]. Locking is implemented by the TMs, in cooperation with the DMs which contain lock managers. In a particular system, TMs and DMs may be specialized computers, as for example: workstations or servers in the client/server model [I8, 3I]. The TM responsible for a transaction's processing issues subtransactions to the appropriate DMs in response to commands from the transaction. Each subtransaction is a sequence of read and/or write requests to the local database. The TM issues two additional requests: commit and abort. Commit tells the DM that: the transaction has terminated, all of its data items should be permanently reflected in the local database, and all of its locks should be released. Abort tells the DM that: the transaction has terminated abnormally, all changes prepared by the transaction should be undone, and all of its locks should be released.

Data lock management is realized by lock managers encapsulated in DMs. For simplicity, we assume that only two basic lock modes are used by a lock manager: read lock mode (LR) and write lock mode (LW). However, rather than setting only read and write locks on data items, DM can exploit the semantics of operations to achieve increased concurrency using type-specific locking [I4, 29].

With each data item x, (i.e., with each page) the DM associates a data lock record, denoted by dlr(x). A data lock record is dynamically created when a data item is first locked, and then destroyed when a data item is unlocked. The structure of a data lock record is shown in Fig. 2.I.

For each transaction T_i, which requests access to data item x, we indicate the transaction identifier and the requested lock mode - T_i:LR(x) or T_i: LW(x). Two transaction lists are associated with each data lock record, namely:

OL - the owner list of the data item lock, and WL - the waiter list for the data item lock. If a read or write request to data item x cannot be granted, due to the incompatibility of locks, it is placed on WL. For each transaction placed on WL, a time-out is defined. If a transaction request from WL cannot be granted within its time-out period, then it has to be aborted. Deadlock and permanent blocking are avoided because aborted transactions release their locks. This solution, however, may lead to either the cyclic or infinite restarting problem.

Let us consider the following example to illustrate the possibility of cyclic restarting phenomenon.

Example I

Let us consider the transaction set $\tau = (T_1, T_2, T_3)$ such that

readset(T_1)= writeset(T_2)= (x),

readset(T_2)= writeset(T_3)= (y),

readset(T_3)= writeset(T_1)= (z).

Let us assume the following concurrent execution of τ:

x: $\ldots\ldots T_1:\underline{LR}(x)\ldots\ldots T_2:\overline{LW}(x)$

y: $\ldots\ldots T_2:\underline{LR}(y)\ldots\ldots T_3:\overline{LW}(y)$

z: $\ldots\ldots T_3:\underline{LR}(z)\ldots\ldots T_1:\overline{LW}(z)$

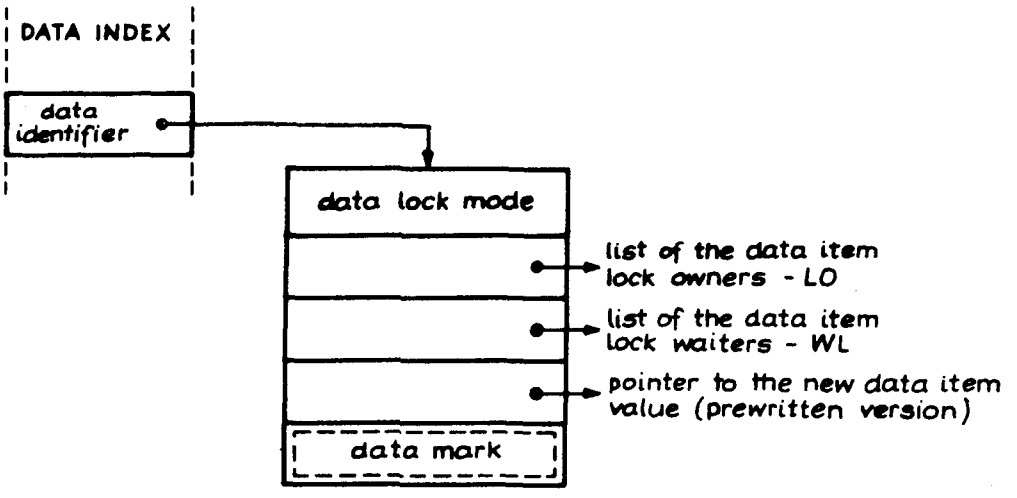

Fig. 2.I. **Data lock record structure**

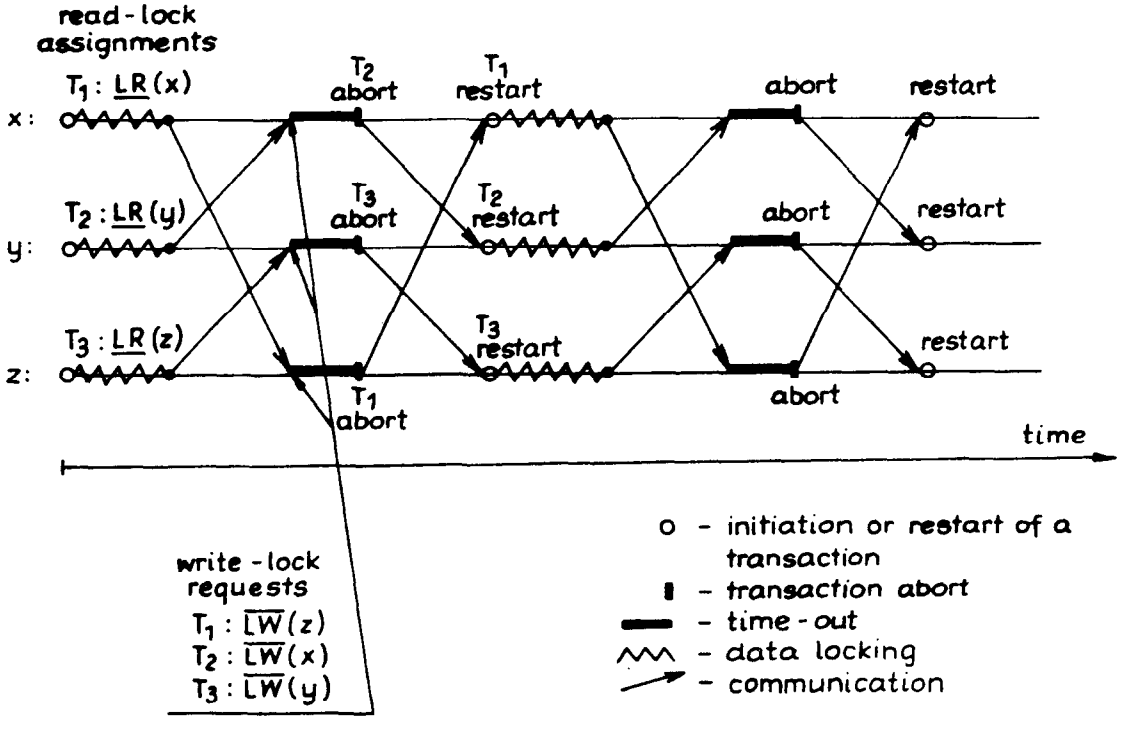

Fig. 2.2. An example of cyclic restarting

where $\overline{LR}(\overline{LW})$ means read(write) lock requests, and $\underline{LR}(\underline{LW})$ means read(write) lock assignment. The above execution leads to deadlock. After the expiration of time-outs, transactions T_1, T_2 and T_3 are rolled back and restarted again. The above situation may possibly repeating forever. This simple example of cyclic restarting of transactions T_1, T_2 and T_3 is illustrated in Fig. 2.2. ■

3. Locking with prevention of cyclic and infinite restarting

We are interested in the design of a cyclic and infinite restarting prevention method in which the control decisions are completely distributed and inter-site communication is maximally limited. In particular, we set the following design goals:

(I) The solution to the cyclic and infinite restarting should not unnecessrily restrict data accessibility. It should also not affect the processing of transactions if the mentioned phenomena do not occur. The solution proposed cannot lead to the deterioration of system performance, (e.g., to the increase of the mean response time).

(2) The proposed solution ought to be easily implemented, particularly in 2PL, and ought to have a low runtime overhead.

Our method fulfills the requirements mentioned above. It involves two modifications to the basic locking mechanism.

The first one consists of introducing the so called data marking mechanism which is applied if a repeatedly restarted transaction has been encountered. We will now describe this mechanism

Each transaction T_i initially receives a unique identifying timestamp: $TS(T_i)$. Transaction timestamps define the total ordering of transactions residing in the system. We assume that the transaction timestamp is kept by a transaction, even if it has been rolled back and restarted.

For each transaction T_i initiated in the system, we define the restart indicator, denoted by $r(T_i)$. If the number of restarts of transaction T_i during its processing exceeds $r(T_i)$. then transaction T_i is considered to be involved in either cyclic or infinite restarting.

With each data lock record dlr(x), we associate a data-mark(x) (cf. Fig. 2.I.). We assume the values of the data marks are equal to $+\infty$ for all no marked data items.

Whenever transaction T_i^* is detected as having restarted repeatedly it marks all the data items necessary for its completion[1]. Data marking consists of the assignment of the transaction T_i^* timestamp $TS(T_i^*)$ to the data mark. Data marking is realized according to the following scheme:

$$\textbf{if } \text{data-mark}(x) > TS(T_i^*) \quad \textbf{then} \quad \text{data-mark}(x) \leftarrow TS(T_i^*) \tag{3.I}$$

The second modification to the basic locking concurrency control algorithm consists of the introduction of the locking precondition, which is called and tested whenever a data item lock is requested.

The locking precondition is defined as follows.

Locking precondition

Transaction T_i is allowed to lock data item x if and only if $TS(T_i) \leq \text{data-mark}(x)$ (3.2)

Note that according to the above definition, an access request of transaction T for data item x can be granted (i.e., T may lock data item x) if and only if both of the following conditions are fulfilled:

(I) $TS(T) \leq \text{data-mark}(x)$;

(2) the lock mode requested for transaction T is compatible with lock mode of dlr(x).

Let the modified 2PL algorithm, with marking as a mechanism of prevention of cyclic and infinite restarting, be denoted by 2PLM. It is fairly easy to prove the correctness of the 2PLM.

Theorem:

The 2PLM is correct in the sense that serializability is guaranteed, and each transaction is completed in finite time.

Proof Sketch:

The basic 2PL is well known to be

---

[1] Marking transaction will be denoted by T_i^*

correct in the sense of serializability [6, 32]. Thus, to prove the first part of the theorem, it suffices to show that 2PLM only permits accesses to data items which would be permitted by 2PL. This fact is obvious and follows immediately from the conditions (I) and (2) presented above.

To prove the second part of the theorem, it is sufficient to show that problems of deadlock, permanent blocking, cyclic and infinite restarting are correctly resolved. Let us note that no deadlock and permanent blocking will last indefinitely, because of the abortion. Cyclic and infinite restarting are resolved by data-marking. The data marking defines a priority scheme, in which the priorities of transactions are defined by their timestamps. This priority scheme guarantees that marked transaction T_i^* with the highest priority (i.e., the oldest one) may reserve all the data items needed for its completion. Thus, after some finite time (and perhaps some restarts needed for the termination or abortion of transactions which currently own the locks of data items marked by transaction T_i^*), T_i^* will be successfully completed. As we assumed, each restarted transaction retains its old timestamp. Thus, the priority of each transaction T_i^* increases with the number of completed transactions and after a finite time, transaction T_i^* reaches the highest priority in the system. Therefore, neither cyclic nor infinite restarting cannot occur in the system. ■

Let us note that the serialization order of transactions for 2PLM is doubly defined. When cyclic and infinite restarting do not occur in the system, and no transaction marks data, the serialization order is determined by the commit points of transactions (i.e., in the same manner as in the 2PL [I, 6,32]). In the case when cyclic and or infinite restarting often occur, the serialization order of transactions is determined by the order of the timestamps of the marked transactions (similarly to the timestamp ordering approach [I]).

Let us consider the following example to illustrate the processing of 2PLM.

Example 2

Let us consider the transaction set τ from Example I. We assume that $TS(T_1) < TS(T_2) < TS(T_3)$. We have shown that concurrent processing of τ may lead to cyclic restarting. Now, we will show how the cyclic restarting from Example I will be resolved by 2PLM.

Whenever the DDBMS detects that the number of T_1, T_2 and T_3 restarts exceed the restart indicator value, then it requires them to mark data (we then denote them as marking transactions T_1^*, T_2^*, T_3^*). During the next restart cycle, transactions T_1^*, T_2^*, T_3^* mark all requested data items.
The following assignments are realized:

$$\text{data-mark}(x) \leftarrow TS(T_1^*)$$
$$\text{data-mark}(y) \leftarrow TS(T_2^*)$$
$$\text{data-mark}(z) \leftarrow TS(T_1^*)$$

Now, after the generation of the same requests as in Example I:

x: .....T_1^*: $\overline{LR}(x)$.....
y: .....T_2^*: $\overline{LR}(y)$.....
z: .....T_3^*: $\overline{LR}(z)$.....

only transactions T_1^* and T_2^* obtain accesses to data items x and y due to the fulfillment of precondition (3.2):

$$TS(T_1^*) \leq \text{data-mark}(x)$$
$$TS(T_2^*) \leq \text{data-mark}(y)$$

Transaction T_3^* does not fulfill the locking precondition (3.2).

Furthermore, let transactions T_1^* and T_2^* generate the write requests: T_1^*: $\overline{LW}(z)$ and T_2^*: $\overline{LW}(x)$. Only the access request of T_1^* is accepted, since $TS(T_1^*) \leq \text{data-mark}(z)$; while the request T_2^*: $\overline{LW}(x)$ is placed on the OL for data item x. After commitment and termination of T_1^*, data item x is unlocked, data-mark(x) is set to $+\infty$ and x may be accessed by T_2^*. Then T_2^* runs to completion and unlocks data item y. Transactions T_1, T_2, T_3 are realized correctly. The serialization order determined by 2PLM is the following:

$$T_1 < T_2 < T_3$$

where < means here the precedence relation. ■

4. Performance evaluation of 2PLM

In this section, the results of the performance evaluation of 2PLM are presented and compared with 2PL. In order to evaluate the performance of 2PLM, a special DDBS simulator was constructed. The entire process of message exchage between the TMs and DMs was

simulated for locking and marking, as described in Section 2.2 and 3.

The main aim of the simulation experiment was to give the answer to the question concerning the reduction of data item accessibility implied by the locking precondition (3.2). We were interested in how the locking precondition (3.2) influenced the DDBS performance.

The following assumptions were made:

(I) the DDBS contained 5 sites;
(2) there were 500 data items uniformly distributed over the DDBS sites[1];
(3) the size of each data item was I0 units, each unit corresponding to transmission packet;
(4) each site concurrently processed 5 transactions, so the degree of parallelism in the DDBS was 25;
(5) references to data items were uniformly distributed over all of them;
(6) the write-set of a transaction was not necessarily contained in its read-set;
(7) there was one restart indicator r, common for all transactions;
(8) transaction size was defined as the number of data items accessed by the transaction over total number of data items in the DDBS;
(9) transaction coefficient of selectivity for a data item x, was defined as the ratio of the size of the result of a transaction operation on x to the size of x, and was assumed to be from the range (0.I - I.0);
(I0) mean packet transmission time between any pair of DDBS sites was constant;
(II) data volume transmission time was proportional to its size;
(I2) message transmission time was equal to packet transmission time;
(I3) the network was assumed to be completely reliable and unbounded (i.e., transmission times did not depend on network loading);
(I4) in order to obtain one point of the curve, at least I00 transactions had to be completed;
(I5) because of 2PL, we only took into account those simulations for which neither cyclic nor infinite restarting occurred ;
(I6) the mean response time to a transaction, and the mean number of transaction restarts were selected as the DDBS performance measures.

Simulation results are shown in Fig. 4.I and Fig. 4.2. Let us first note that the performance of both 2PL and 2PLM heavily depends on transaction size. It follows from the fact that the increase of the mean transaction size cause the considerable increase in the number of lock incompatibility conflicts between transactions. Thus, a very significant increase in the mean response time and mean number of restarts versus transaction size are observed in Fig. 4.I and Fig. 4.2.[2]

As it can be seen from Fig. 4.I and Fig. 4.2, for small mean transaction size, the performance of 2PLM is the same as that of 2PL. It means that for small mean transaction size, the data marking mechanism does not affect the processing of transactions, which are processed strictly according to 2PL. On the other hand, the performance of both algorithms differs considerably with increasing mean transaction size, followed by the increasing number of transaction conflicts. This means that the data marking mechanism starts to schedule transactions.

As can be seen from the presented curves, the performance of 2PLM depends heavily on the choice of the restart indicator value. The appropriate choice of the restart indicator value may significantly improve the efficiency of 2PLM, which then provides higher performance than 2PL. It means that data marking ensures not only the correct resolution of cyclic and infinite restarting problems, but may also considerably improve the overall DDBS performance, despite some data item accessibility restrictions. This fact can be explained in the following way.

In a DDBS running under 2PL, a number of transactions can be involved in cyclic or infinite restarting. These transactions are characterized by an excessive number of restarts and a long response time. Cutting off the cycle of transaction restarting early, and then executing them strictly according to their timestamps, we significantly reduce their response times. This con-

---

[1] Let us explain that the number of data items was restricted to 500 because the phenomena examined here concerns only frequently accessed data items as for example directories,indices etc., which are not numerous in real DDBSs.

[2] Similar results, regarding 2PL, were obtained by Gray [7,8]. Gray asserts that transaction conflicts are a function of the square of the mean transaction size.

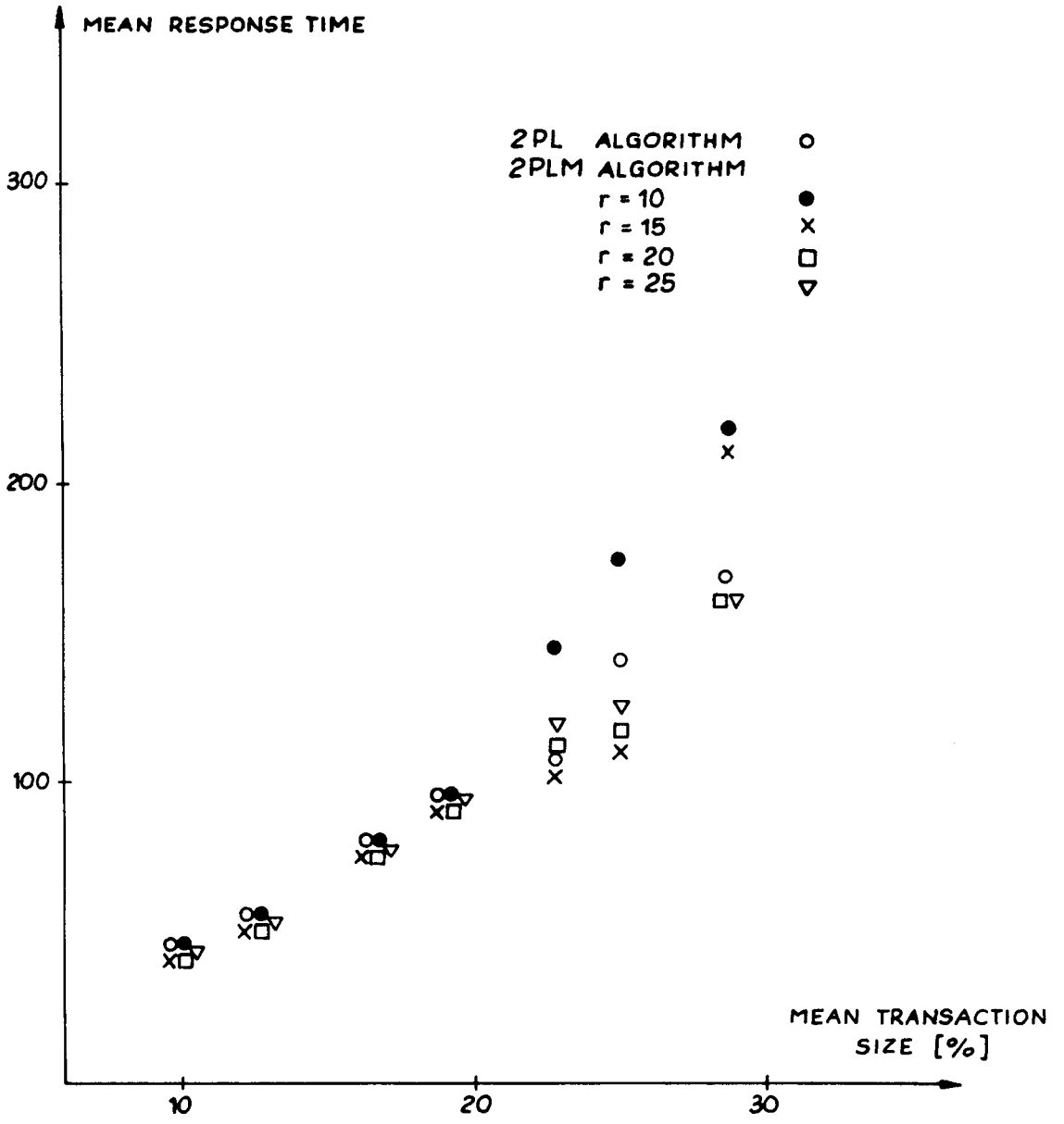

Fig. 4.I. Mean response time for 2PLM and 2PL algorithms versus transaction size for different values of the restart indicator.

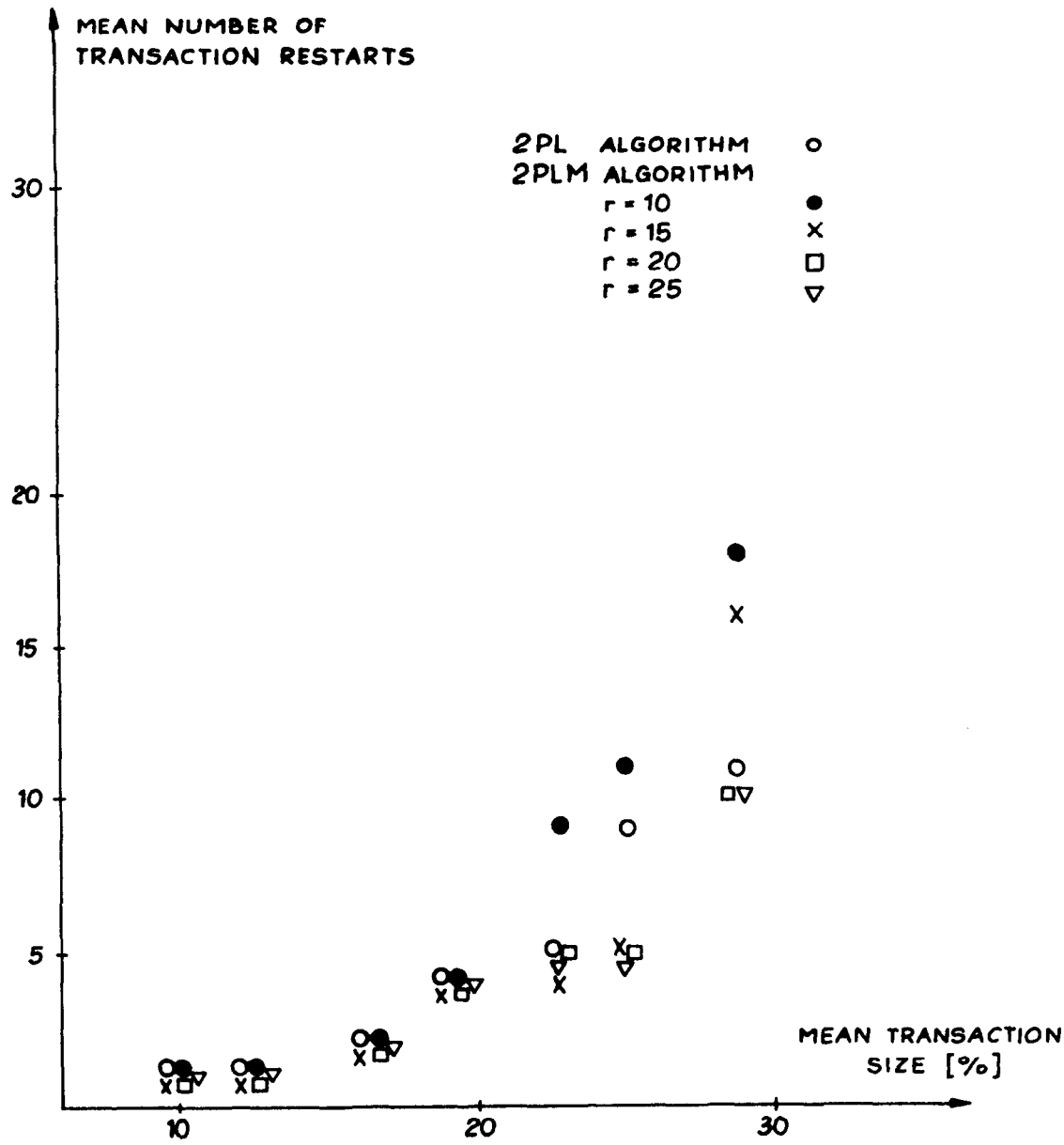

Fig. 4.2. Mean number of transaction restarts for 2PLM and 2PL algorithms versus transaction size for different values of the restart indicator.

tributes to the improvement of the mean DDBS response time.

Of course, a trade off occurs between the reduction of the response times of repeatedly restarted transactions and the restriction of some data items accessibility, which is followed by some increase in restart numbers and the response times of other transactions.

If, for a given system loading, the restart indicator value is too small, then the number of transactions marking data becomes high and data accessibility is severely restricted. The level of concurrency in the system then decreases unnecessarily, because many transactions are processed sequentially.

On the other hand, if the restart indicator value is too great, then both the suppression of repeated restarting and the completion of the transactions concerned is unnecessarily delayed.

An interesting relationship between the appropriate restart indicator value for 2PLM and the mean number of transaction restarts in a DDBS using 2PL can be drawn from the detailed analysis of the simulation results presented. It appears that for a given system loading, a nearly maximal performance is obtained when the restart indicator is set to the mean number of transaction restarts in a DDBS running under 2PL.

5. Conclusions

The solution to the cyclic and infinite restarting problems proposed in this paper fulfills the design requirements stated in Section 3. In particular, it affects the processing of transactions only in the case of a real danger of cyclic or infinite restarting, and can be easily implemented in DDBSs. Moreover, because of the reduction of the total number of transaction restarts, the overall DDBS performance is improved.

Finally, we would like to point out that the infinite restarting problem concerns not only the locking approach to the concurrency control in DDBSs, but also the validation and timestamp ordering approaches [I, 5,I3, I6, 25, 26, 3I]. A prevention method similar to that presented in this paper can be used to solve this problem.

References

I P.A. Bernstein, N. Goodman,"Concurrency control in distributed database systems",Computing Surveys, vol. I3, no. 2, I98I, pp. I85-22I

2 P. Bouchet, A. Chesnais, J.M. Feuvre, G. Jomier, A. Kurinckx," PEPIN: An experimental multimicrocomputer data base management system", Proc. 2nd Int. Conf. on Distributed Computing Systems, Paris, I98I, pp. 2II-2I7

3 M.J. Carey,"Granularity hierarchies in concurrency control", Proc. ACM SIGACT-SIGMOD Symp. on Principle of Database Systems, Atlanta. I983,pp. I56-I65

4 W. Cellary,"Resource allocation in computer systems - an attempt at a global approach", Wydawnictwo Politechniki Poznanskiej. Poznan. I98I (in polish)

5 S. Ceri, S. Owicki,"On the use of optimistic methods for concurrency control in distributed databases", Proc. 6th Berkeley Workshop on Distributed Data Management and Computer Networks, Berkeley, I982

6 K.P. Eswaran, J.N. Gray, R.A.Lorie I.L. Traiger,"The notion of consistency and predicate locks in a database system", Comm. ACM, vol. I9, no. II, I976, pp. 624-633

7 J. Gray,"A transaction model", IBM Res. Rep. RJ 2895, I980

8 J. Gray."Transaction concept: virtues and limitations", Proc. 7th Int. Conf. on Very Large Data Bases, Cannes, I98I,pp. I44-I54

9 V.D. Gligor, S.H. Shattuck,"On deadlock detection in distributed systems", IEEE Trans. Software Eng., vol. SE-6, no. 5, I980,pp. 435-440

I0 G.S. Ho, C.V. Ramamoorthy, "Protocols for deadlock detection in distributed database systems", IEEE Trans. Software Eng., vol. SE-8, no. 6, I982,pp. 554-557

II S.S. Isloor, T.A. Marsland,"The deadlock problem: an overview", IEEE Computer. I980,pp.58-78

I2 J.R. Jagannatan, R. Vasudevan, "Comments on protocols for deadlock detection in distributed database systems", IEEE Trans. Software Eng., vol. SE-9, no. 3, I983,pp. 37I-37I

I3 W.H. Kohler,"A survey of techniques for synchronization and recovery in decentralized computer systems", Computing Surveys, vol.

I3, no. 2, I98I,pp. I49-I83
I4 H.F. Korth, "Locking primitives in a database system", Journal of the ACM, vol. 30, no. I, I983
I5 H.T. Kung, C.H. Papadimitriou, "An optimality theory of database concurrency control", Proc. ACM SIGMOD Int. Conf. on Management of Data, I979,pp.II6-I26
I6 H.T. Kung, J.T. Robinson, "On optimistic methods for concurrency control", ACM Trans. Database Syst., vol. 6, no. 2, I98I, pp. 2I3-226
I7 D.A. Menasce, R.R. Muntz, "Locking and deadlock detection in distributed data bases", IEEE Trans. Software Eng., vol. SE-5, no. 3, I979,pp. I95-20I
I8 J.G. Mitchell, J. Dion, "A comparison of two network based file servers", Comm. ACM, vol. 25, no. 4, I982,pp.233-245
I9 T. Morzy, "Ordered-transaction approach to performance evaluation of concurrency control algorithms for distributed database systems", Proc. Int. Conf. on Management of Distributed Data Processing, Paris, I982,pp. 253-269
20 T. Morzy, "Concurrency control in distributed database systems", Ph. D. thesis, Inst. of Control Eng., Technical Univ. of Poznan, I983 (in polish)
2I J.E.B. Moss, "Nested transactions an approach to reliable distributed computing", Ph. D. thesis, MIT, LCS/TR-260, I98I
22 R. Obermack, "Global deadlock detection algorithm", IBM Res. Rep. RJ 2845, I980
23 C.H. Papadimitriou, "The serializability of concurrent database updates", Journal of the ACM, vol. 26, no. 4, I979,pp.63I-653
24 C.H. Papadimitriou, P. Kanellakis "On concurrency control by multiple versions", Proc. ACM SIGACT-SIGMOD Symp. on Principle of Database Systems, I982, pp. 76-82
25 D.P. Reed, "Naming and synchronization in a decentralized computer system", Ph. D. thesis, MIT LCS/TR-205, I978
26 J.T. Robinson, "Design of concurrency control for transaction processing systems", Ph. D. thesis, Computer Sc. Dept. Carnegie-Mellon, Pittsburgh,I982
27 D.J. Rosenkrantz, R.E. Stearns, P.M. Lewis, "System level concurrency control for distributed database systems", ACM Trans. Database Syst., vol. I, no. 2, I978,pp.I78-I98
28 G. Schlageter, "Optimistic methods for concurrency control in distributed database systems", Proc. 7th Int. Conf. on Very Large Data Bases, Cannes, I98I,pp. I25-I30
29 P.M. Schwarz, A.Z. Spector, "Synchronizing shared abstract types", Tech. Rep. CMU-CS-83-I63, I983
30 M. Stonebraker, "Concurrency control and consistency of multiple copies of data in distributed INGRES", IEEE Trans. Software Eng., vol. SE-5, no. 3, I979,pp. I88-I94
3I L. Svobodova, "File servers for network-based distributed systems" Tech. Rep. IBM Zurich, Ruschlikon, RZ II87, I982
32 M. Yannakakis, "Serializability by locking", Journal of the ACM, vol. 30, no. 2, I984,pp. 227-244