Title	Controllable Two-Phase Locking Mechanisms
Author(s)	KAMBAYASHI, Yahiko
Citation	数理解析研究所講究録 (1988), 655: 40-51
Issue Date	1988-04
URL	http://hdl.handle.net/2433/100515
Right	
Type	Departmental Bulletin Paper
Textversion	publisher

数理解析研究所講究録
第655巻 1988年 40-51

Controllable Two-Phase Locking Mechanisms

可制御二相施錠機構

Yahiko KAMBAYASHI
上林弥彦
Dept. of Computer Science and Communication Eng.
Kyushu University
九州大学工学部
Fukuoka 812,Japan

The concept of controllability of concurrency control mechanisms was introduced by the author. It is useful to realize a distributed system consisting of different kinds of concurrency control mechanisms. We can also realize adaptive concurrency control mechanisms as well as mechanisms to handle complex data which are mutually related. This paper discusses a method to realize controllable two-phase locking mechanisms. In a controllable concurrency control mechanism, from outside we can add order information on transactions and the mechanism produces the serializable schedule which has no conflicts with the given order. A wait-for graph with control edges is introduced to realize controllability by modifying the lock phase of two-phase locking mechanisms. We will show conditions when rollback of a transaction is required using that graph in order to satisfy the given order.

1. Introduction

It is important to use concurrency control mechanisms to improve efficiency of database systems. There are many such mechanisms proposed so far and each of which has different advantages. We need to develop concurrency control mechanisms suitable for wide range of applications. For such purpose the author has introduced the concepts of controllability and observability of concurrency control mechanisms. In this paper we will discuss how to make one of the typical mechanisms, two-phase locking mechanisms, to be controllable.

A concurrent execution of transactions is defined to be correct if the result of the execution is equivalent to the result of some serial execution of transactions. In conventional concurrency control mechanisms, we cannot select the serial execution equivalent to the concurrent execution. If a concurrency control mechanism is controllable, some restrictions can be imposed on such serial schedules from outside. If it is observable, such equivalent serial orders can be known from outside. Such concept was introduced in order to combine database systems with different concurrency control mechanisms [KAMB85].

There are the following applications of such properties.

(1)Dynamic concurrency control mechanisms : We can control the property of the concurrency control mechanism according to the change of usage patterns.

(2)Combination of different concurrency control mechanisms : It is not possible to combine two database systems with different concurrency control mechanisms, for example two-phase locking and time-stamp ordering mechanisms. One simple method is to make one mechanism to be observable and the other to be controllable, so that the serial schedule observed from one system is used to control the other system in order to avoid inconsistency.

(3)Priority control : For some applications we need to classify transactions by their priority. Priority control can be realized by controllable concurrency control.

(4)Handling of non-uniform transactions : In time-stamp ordering mechanisms, among conflicting transactions always the transaction started earlier than others is selected to be restarted. Thus if there are transactions longer than others, the possibility of rollbacks of these transactions is higher and we may not be able to finish very long transactions. We need to control such transactions not to be restarted.

There are two typical concurrency control mechanisms; two-phase locking and time-stamp ordering ones. As control of time-stamp ordering mechanisms is discussed in [KAMBZ87] together with applications to problem (4) above, in this paper we will discuss methods to make two-phase locking mechanisms to be controllable. Control can be realized by adding control edges to wait-for graphs which are used to detect conflicts in two-phase locking mechanisms.

Basic concepts will be discussed in Section 2. Section 3 shows definitions related to controllable concurrency control mechanisms. How to realize controllable two-phase locking mechanism is discussed in Section 4. For simplicity we will discuss strict two-phase locking with only execlusive locks.

2. Basic Concepts

Let $A_i, B_i, \ldots$ be units of data handled by read and write operations. A transaction is assumed to be expressed by a sequence of read and write operations. A read or write operation for data item A_i performed by transaction T_j is denoted by $R_j(A_i)$ or $W_j(A_i)$, respectively. A schedule S for a set of

transactions $\{T_1,T_2,...,T_m\}$ satisfies the following condition, where f_i is an operation to erase all operations R_j and W_j when $j \neq i$.

$$f_i(s) = T_i$$

A schedule S is serial if it is expressed by a sequence of T_{ai}'s, where (a1,a2,...,am) is a permutation of (1,2,...,m). Two schedules are said to be equivalent if for any combination of initial data values the outputs of transactions and the final values of all the data after the execution by the both schedules are identical. A schedule is called serializable if it is equivalent to some serial schedule. A serializable schedule is assumed to be a correct schedule. Typical methods to generate serializable schedules are two-phase locking mechanisms and timestamp ordering mechanisms. We will discuss two-phase locking mechanisms in this paper. In order to avoid a series of rollback operations the following strict two-phase locking mechanism is usually used.

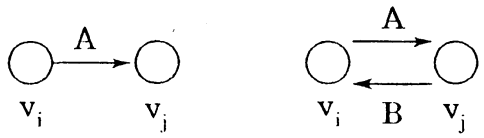
[Definition 1] Strict Two-Phase Locking Mechanism
Every transaction consists of the following three steps.
(1)Lock phase : During computation when a data item is required, a lock request for the data item is issued. If the data item is not locked by another transaction, it can be locked, otherwise the transaction must wait until the data item becomes available. Lock operations and computation are realized in this phase.
(2)Computation step : After locking all the data items required by the transaction, only computation is performed.
(3)Unlock phase : After the completion of computation all locked data items are unlocked at the same time.

Data items once locked in the lock phase will be never unlocked before the unlock phase. If a transaction T_i issues a lock request to data item A in the lock phase and it is detected that A is already locked by another transaction T_j, T_i must wait until A is unlocked by T_j. Such wait conditions are not simple, since T_j

may also wait for another data item to be unlocked. The following wait-for graph is used to show the interaction of lock requests.

[Definition 2] Wait-for Graph

A wait-for graph G is a labeled directed graph such that V is a set of vertices, E is a set of edges and L is a set of labels. Each vertex corresponds to a transaction and each edge (called a wait-for edge) corresponds to a lock request. If transaction T_i made a request for data item A which is locked by transaction T_j, there is a direct edge labeled by A from v_i to v_j as shown in Fig.1(a).



(a) (b)

Fig. 1 Basic components of a wait-for graph

If T_i makes a lock request for data item A locked by T_j and T_j makes a lock request for data item B locked by T_i, the both transactions will wait forever. Such a situation is called a deadlock. The wait-for graph for this case is shown in Fig.1(b). In this case if T_i or T_j unlocks all the data items the deadlock will be eliminated. In order to unlock all the data items the transaction must be restarted from the beginning. Such an operation is called a rollback operation. In general if there is a loop in the wait-for graph as shown in Fig.2(d) then there is a deadlock. We must select one transaction in the loop to be rollbacked in order to delete the loop.

In the two-phase locking mechanism, the transactions which correspond to vertices without outgoing edges are active and the transactions correspond to

vertices with outgoing edges are in waiting state. Active transactions are shown by black circles in Fig.2.

The wait-for graph is modified by termination of a transaction and generation of new lock requests. Fig.2(a) shows a situation when there are two transactions waiting for the transaction shown by a black circle. After the termination of this transaction one of the two waiting transactions locks data item A and the graph in Fig.2(b) is obtained. If the transaction shown by a black circle in Fig.2(a) makes a new lock request, there are two cases, nondeadlock and deadlock cases, which are shown in Fig.2(c) and (d), respectively. In Fig.2(c) the transaction becomes waiting state. Fig.2(d) contains a loop corresponding to a deadlock, due to the lock request shown by the dotted line.

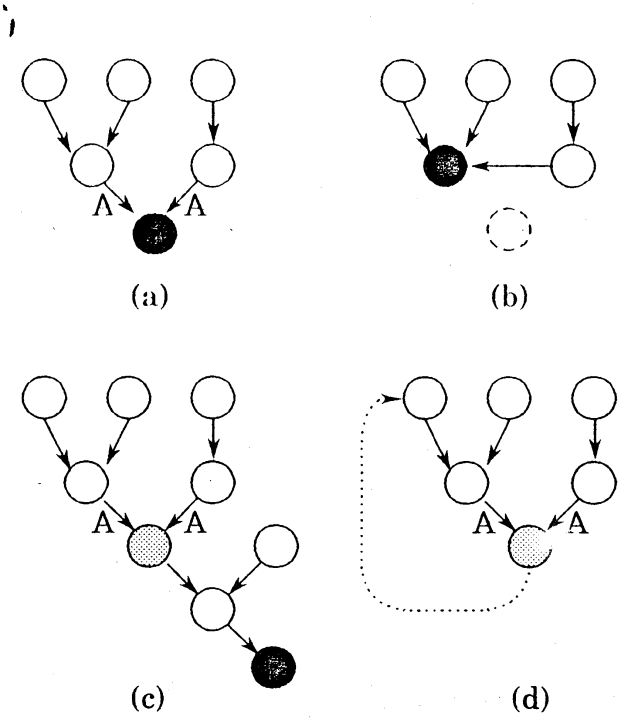


Fig.2 Modification of wait-for graphs

3. Controllability of Concurrency Control Mechanisms

In this section definitions related to controllable concurrency control mechanisms are discussed.

Let P be a partial ordered set on the set of transactions. Each element of P is denoted by $T_i > T_j$.

[Definition 3] Let P be a partial ordered set. A closure of P, denoted by P*, is generated by the application of the following operation recursively until no new elements are generated.

If there are $T_i > T_j$ and $T_j > T_k$ in P, add $T_i > T_k$ which is generated by the transitivity rule on partial ordered set.

[Definition 4] Let P_i and P_j be two partial orders. If every element in P_j is contained in P_i*, P_i is said to be a refinement of P_j.

[Definition 5] Let P_i and P_j be two partial orders. If there are no conflicting orders in P_i* and P_j*, P_i and P_j are said to be conflict-free. Here it is said to be conflicting orders if P_i* contains $T_k < T_h$ and P_j* contains $T_k > T_h$.

Orders of read and write operations define orders of transactions. For example, if there is a sequence $W_j(A)R_k(A)$ in schedule S then there is $T_j < T_k$ in the partial order realized by S, since data A written by T_j is read by T_k. The partial order defined by the two-phase lock mechanism is as follows.

[Definition 6] The partial order P defined by the two-phase lock mechanism consists of the following elements.

$T_i < T_j$ if T_i and T_j lock the same data item and T_j locks the data item after unlocked by T_i.

If we define shared and exclusive locks corresponding to read and write operations, the efficiency of the mechanism will be increased. For simplicity we will only discuss (exclusive) locks in this paper.

[Definition 7] A concurrency control mechanism is controllable if the mechanism can generate the partial order P for any given partial order Q, such that P and Q are conflict-free.

[Definition 8] A concurrency control mechanism is weakly controllable if it is controllable for some restricted class of partial order Q.

If we can only use total order as Q, it is weakly controllable. As two-phase lock mechanisms are not controllable we will discuss a method to make it to be controllable.

4. Controllable Two-Phase Locking Mechanisms

In order to realize a controllable two-phase locking mechanism, a wait-for graph with control edges is defined as follows.

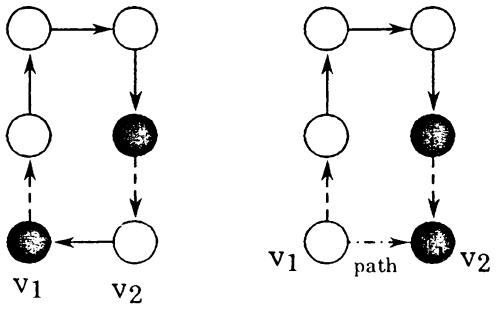
[Definition 9] A wait-for graph Gc with control edges is defined by Gc =(V,E,F,L), where V is a set of vertices, E is a set of wait-for edges, F is a set of control edges, and L is a set of labels for E. F is defined to control the order realized by a serializable schedule. There exists control edge f_{ij} from v_i to v_j, when there is element $T_i < T_j$ in Q which is given as controlling partial order.

We use dotted lines to express control edges. In order to realize controllable mechanism, we need to control a wait-for graph with control edges to be loop-free.

[Theorem 1] If a wait-for graph with control edges has a loop consisting of at least one control edge, the order of execution will eventually have a conflict with the order Q which is given as a control input.

Proof : By properties of wait-for graphs with control edges, the vertex corresponding to the transaction which is active has no outgoing edges and possibly outgoing control edges. In a wait-for graph with control edges a loop with at least one control edge does not produce a deadlock. Fig.3(a) shows such a

situation. Here the transactions shown by black circles are active. In this case if all transactions in the loop are executed without rollbacks, the order realized by the mechanism has a conflict with the order determined by the control edges.



(a) (b)

Fig.3 A loop in a wait-for graph with control edges

[Theorem 2] If there is a loop in a wait-for graph with control edges, we can eliminate the loop by rollbacking a transaction which corresponds to the vertex whose incoming edge in the loop is not a control edge.

Proof : We can change the directions of edges in E by rollback operations, but the direction of control edges cannot be changed. Thus we have to rollback a transaction whose incoming edge is not a control edge. In Fig.3(a), if the transaction corresponding to v_1 is rollbacked (the data items locked by v_1 are unlocked), then direction of the edge between v_1 and v_2 can be changed effectively (in general there is a path from v_1 to v_2 as shown in Fig.3(b)) and thus the loop is deleted. Here by the roll back operation, the transaction corresponding to v_1 starts from the beginning, and thus v_1 will use the data item after unlocked by the transaction corresponding to v_2.

If there is a loop consisting of only wait-for edges, we must select one transaction to be rollbacked. If a loop contains at least one control edge, we need not rollback one transaction immediately, since there is at least one active transaction in the loop. If there is more than one such loop we can determine a

minimum set of transactions to be rollbacked to eliminate all the loops. Since Q for control is a partial ordered set, there is no loop consisting of only control edges.

Even if there is no loop currently, by improper operations we may generate a loop. How to avoid such possible loops is discussed below.

[Theorem 3] Assume that there are the following v_0, v_1 and v_2.

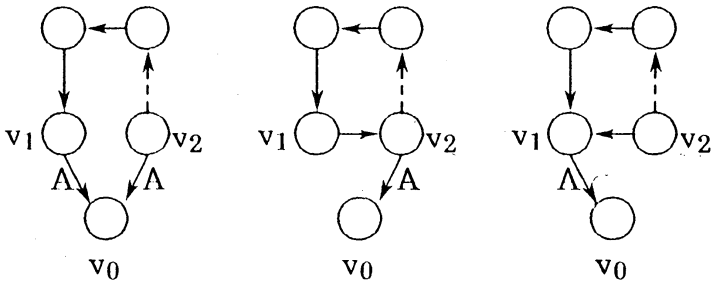
There are edges e_{10} and e_{20}.

The labels of the two edges are identical.

There is a path from v_2 to v_1 consisting of edges in both E and F.

In this case after the completion of the transaction corresponding to v_0, the right to lock the data should be given to the transaction corresponding to v_1 to avoid a potential loop.

Proof : A graph satisfying the above conditions is shown in Fig.4(a). Assume



(a) (b) (c)

Fig.4 Deletion of a possible loop

that the path from v_2 to v_1 contains a control edge. After the completion of the transaction corresponding to v_0, if the transaction corresponding to v_2 gets the right to lock A then there is a loop as shown in Fig.4(b). If the right to lock A is given to the transaction corresponding to v_1, such a loop can be avoided.

In general when there is more than one transaction waiting for the same data item, the order to give the right to the transactions should not have a conflict with the order determined by the edges of the wait-for graph with control edges.

[Theorem 4] If there exists edge f_{ij}, we cannot complete the transaction corresponding to v_i before the completion of the locking phase of the transaction corresponding to v_j.

Proof : By edge f_{ij}, it is required that the transaction corresponding to v_j should be completed before the completion of transaction corresponding to v_i. If v_j is in locking phase there is a possibility that v_j will make lock request for a data item locked by v_i. In such a case the transaction corresponding to v_i must be rollbacked. If v_i is already completed, we cannot rollback v_i and thus v_j cannot be completed. Furthermore, if there is a path which goes into v_i, there are no additional problems since the transactions in the path cannot be completed before the completion of the transaction corresponding to v_i.

By the above discussion we have the following controllable strict two-phase locking mechanism.

[Controllable Strict Two-Phase Locking Mechanism]

(1)Lock phase : If there exists a loop of Theorem 1 in a wait-for graph with wait-for and control edges, a loop is eliminated by rolling back a proper transaction determined by Theorem 2. When there is more than one transaction requesting lock for the same data item currently locked by some transaction, we can avoid possible deadlocks by giving a proper order on locking to the requesting transactions (Theorem 3).

(2)(3) Computation step and unlock phase : Same as the strict two-phase locking mechanism.

5. Concluding Remarks

In this paper how to realize controllable two-phase locking mechanism is discussed. As shown in Section 1 there are many application areas for controllable concurrency control mechanisms, such mechanisms will be

important in complicated systems like distributed system and multi-media systems. We can get an efficient controllable two-phase locking mechanism by considering shared locks as well as exclusive locks.

References

[BERNG80] P. A. Bernstein and N. Goodman "Timestamp-Based Algorithms for Concurrency Control in Distributed Database Systems," Proc. 6th Int. Conf. VLDB, Oct. 1980.

[ESWAG76] K. P. Eswaran, J. N. Gray, R. A. Lorie and I. L. Traiger, "The Notions of Consistency and Predicate Locks in a Database System," Comm. ACM, pp.624-633, 1976.

[KAMB85] Y.Kambayashi,"Concurrency Control in Distributed Database Systems," Symposium on Knowledge Information Processing, Information Processing Society of Japan, September 1985 (in Japanese).

[KAMBK84] Y. Kambaysahi and S. Kondoh, "Global Concurrency Control Mechanisms for a Local Network Consisting of Systems without Concurrency Control Capacity," AFIPS NCC Vol.53, pp31-39, 1984.

[KAMBZ87] Y. Kambayashi and X. Zhong, "Controllable Timestamp Ordering and Oriental Timestamp Ordering Concurrency Control Mechanisms," Proc.IEEE COMPSAC, Oct. 1987.

[PAPA 86] C. H. Papadimitriou, *The Teory of Database Concurrency Control*, Computer Science Press, 1986.

[SAISK87] K.Saisho and Y.Kambayashi, "Multi-Wait Two-Phase Locking Mechanism and Its Hardware Implementation," Proc. International Workshop on Database Machines, Oct. 1987.