PREVENTION OF IMPLICIT DEADLOCK BY OBJECT REALLOCATION

6H−5

Akiko HYOUDOH and Makoto TAKIZAWA
Tokyo Denki University

1. INTRODUCTION

Most database systems and distributed database systems have adopted locking schemes as the synchronization method of interleaved and concurrent executions of transactions. Problem is how to resolve deadlock. In the conventional systems, a deadlock manager detects a deadlock by constructing a wait-for graph and finding a directed cycle in the wait-for graph. The deadlock is resolved by aborting a process in the deadlock cycle. If the objects held by the selected process are arbitrarily allocated to the other process, another deadlock may occur since the other process which has waited on object. We say such deadlock which is caused by aborting the selected process an implicit deadlock. In this paper, we discuss not only how to detect the deadlock and select a process in a deadlock cycle but also how to allocate the objects held by the selected process to the processes which waited on them so as not to cause the implicit deadlock.

2. DEADLOCK

Suppose that the system has a deadlock manager which grasps the systems state, detects a deadlock, and then resolves it. In this paper, we assume that the deadlock manager can obtain a consistent system state by some mechanism presented in [KNAP87]. A wait-for graph for the state S is defined as follows.

[Def.] A wait-for graph Gs for a system state S is a directed graph (Vs, Es) where

(1) Vs is a set of nodes each of which denotes a process in the system, i.e. Vs = $\{P_1, \ldots, P_n\}$, and

(2) Es is a set of directed edges, which includes a directed edge $P_j \rightarrow_x P_k$ for every two different processes P_j and P_k in Vs if P_j waits on an object x held by P_k in S.□

For a process P, let Obj(G,P) be a set of objects which P holds. In the one-resource model and AND model [KNAP87], a consistent system state S is deadlocked if and only if (iff) the wait-for graph Gs for S contains a directed cycle.

[Def.] Let Gs be a wait-for graph (Vs, Es) for a system state S. Let P and Q be a process in Gs. P is said to depend on Q (or Q is reachable from P) in Gs (written as $P \Rightarrow_{Gs} Q$) iff

(1) for every object x in Obj(Gs, Q), $P \rightarrow_x Q$ in Gs, or

(2) there exists some process R in Vs such that $P \Rightarrow_{Gs} R$ and $R \Rightarrow_{Gs} Q$.□

$P \rightarrow Q$ means that a process P depends on Q for some object in Gs.

[Def.] For a wait-for graph Gs, a process P is said to be deadlocked in S iff P depends on itself or depends on some deadlocked process in Gs. P is said to directly deadlocked iff P depends on itself, i.e. $P \Rightarrow_{Gs} P$.□

Directly deadlocked processes are processes which are included in some deadlock cycle. Deadlocked processes which are not directly deadlocked are said to be indirectly deadlocked.

3. IMPLICIT DEADLOCK

In this paper, we consider the AND model [KNAP87] where processes can request more than one object and block until all of them are obtained. Let P_j and P_k be processes, and x be an object in the system. Let S be a consistent system state and Gs be a wait-for graph of S. Suppose that S is deadlocked. Since the deadlock manager is assumed to be able to obtain S, it constructs a wait-for graph Gs for S.

Wait(Gs, P, x) = a set of processes which wait on an object x held by P_j.

WaitP(Gs, P_j) = a set of processes which wait for the objects held by P_j.

DLP(Gs) = a set of directly deadlocked processes in Gs.

LP(Gs) = a set of deadlocked processes in Gs

[Def.] Suppose that a system state S is changed to T by aborting a process P and allocating the objects. Let Gs and G_T be wait-for graphs for S and T, respectively. Let ILP(Gs,P) be a set LP(G_T) − LP(Gs − {P}). Processes in ILP(Gs,P) are said to be implicitly deadlocked.□

(a) G_{1a} (b) G_{1b}

Fig.1 Wait-for Graph

Let us consider a wait-for graph G_{1a} in Fig.1(a). Suppose that a process P is directly deadlocked. When P is selected and aborted by the deadlock manager, a object x held by P is released and is allocated to a process which has waited on it. If x is allocated to Q, a new deadlock cycle, i.e. Q→R→Q appears as shown in Fig.1(b). ILP(G_{1a},P) = LP(G_{1b}) − LP(G_{1a} − {P}) = {Q,R}. Processes Q and R in

PREVENTION OF IMPLICIT DEADLOCK
BY OBJECT REALLOCATION
Akiko HYOUDOH and Makoto TAKIZAWA
Tokyo Denki University

ILP(G_{1a},P) are implicitly deadlocked.
Processes in ILP(Gs,P) are processes which are still deadlocked after the abortion of P. However, the deadlock manager considers them not to be deadlocked. In this paper, we would like to present a deadlock resolution method to prevent implicit deadlocks from occurring.
[Déf.] Let P be a deadlocked process in a wait-for graph Gs. Let RLP(Gs, P) be a set LP(Gs) - LP(Gs - {P}) - {P}. Let ULP(Gs, P) be a set LP(Gs - {P}). Processes in RLP(Gs) are said to be ones which are resolvably deadlocked for P in Gs. Processes in ULP(Gs, P) are said to be ones which are unresolvably deadlocked for P in Gs.□
A family {ULP(Gs,P), RLP(Gs,P), {P}} is a partition of LP(Gs), i.e. LP(Gs) = ULP(Gs,P)∪ RLP(Gs,P)∪{P} and they are pairwise disjoint [Fig.2].

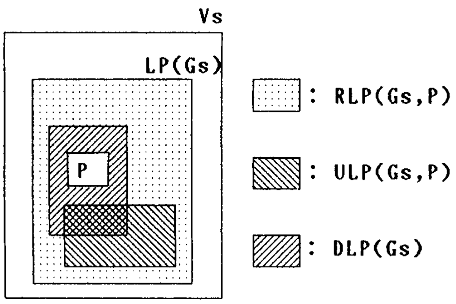

Fig.2 Relationships among deadlock states

There are two kinds of implicit deadlocks, i.e. directly and indirectly implicit deadlocks.
[Def.] Suppose that a system state is changed from S to T by aborting a process P. Let Gs and G_T be wait-for graphs for system states S and T, respectively. If there exists a deadlock cycle C_T in G_T which includes only processes in WaitP(Gs, P) and at least one process in C_T is not in ULP(Gs, P), then processes in C_T are said to be directly implicitly deadlocked in G_T. Implicitly deadlocked processes in G_T which are not directly implicitly deadlocked are said to be indirectly implicitly deadlocked processes.□
[Def.] Let P be a process and x be an object held by P in a wait-for graph Gs, i.e. x∈ Obj(Gs, P). Let G' be Gs - {P}. A process Q∈ Wait(Gs, P, x) is said to be a candidate of x in Gs iff Q is not deadlocked in a wait-for graph G', i.e. Q is not in LP(G'), and Q does not depend on any processes in WaitP(Gs, P) in G'.□
Let Cand(Gs, P, x) be a set of processes which are candidates of x for a process P in Gs. For every object x held by a process, if there exists a candidate of x, we can prevent any indirectly implicit deadlock from occurring by allocating x to its candidate and aborting the process. However, for some object x, unless there exists a candidate of x, deadlocks still exist after the process is aborted.
[Prop. 3.1] For every object x held by a process P in a wait-for graph Gs, if there is no candidate of x for P, every process in Wait(Gs, P, x) is unresolvably deadlocked, i.e. in ULP(Gs, P).
[Proof] From the definition of the candidate process of x for P, if there is no candidate of x, every process in Wait(Gs, P, x) depends on a process in Wait(G', P, x) or in DLP(Gs) in a wait-for graph G' = Gs - {P}. Therefore, even if P is aborted, processes in Wait(Gs, P, x) are still deadlocked.■

4. SELECTION AND ALLOCATION ALGORITHM

As stated before, assume that a global consistent state is already held by the deadlock manager. We show our algorithm for selecting a process and allocating objects obtained by the selected process to the other processes which have waited for them.
[Selection and Allocation Algorithm (SAA)]
(1) Let Gs be a wait-for graph for a consistent system state S.
(2) If LP(Gs) = ϕ, i.e. no deadlock cycle in Gs, then the SAA terminates.
(3) [Deadlock exists, i.e. LP(Gs)$\neq \phi$] Select a process P in DLP(Gs), i.e. directly deadlocked process.
(4) For every object x in Obj(Gs, P),
  (4-1) if there exists a candidate process of x for P, i.e. Cand(Gs, P, x)$\neq \phi$, select a candidate process Q in Cand(Gs, P, x),
  (4-2) else [There is no candidate of x] select a process Q in Wait(Gs, P, x),
  (4-3) allocate x to Q, and Gs := Gs + { U $\rightarrow_x$ Q ¦ U∈(Wait(Gs, P, x) - {Q}) } - { U $\rightarrow_x$ P ¦ U∈Wait(Gs, P, x) }.
(5) Abort P and Gs := Gs - {P}. Go to (2).□
[Theorem] Our SAA algorithm never causes implicit deadlock.■

5. CONCLUDING REMARKS

In this paper, we have presented how to prevent implicit deadlock implied by aborting the selected process. By allocating the objects held by the selected process to the candidate processes which wait for the objects, the implicit deadlock is prevented by our algorithm.

REFERENCES

[CHAN83] Chandy, K. M., Misra, J., and Haas, L. M., "Distributed Deadlock Detection," ACM TODS, Vol.1, No.2, 1983, pp.144-156.
[KNAP87] Knapp, E., "Deadlock Detection in Distributed Databases," ACM Computing Surveys, Vol.19, No.4, 1987, pp.303-328.
[LEIB89] Leibfried Jr., T. F., "A Deadlock Detection and Recovery Algorithm Using the Formalism of a Directed Graph Matrix," ACM Operating Systems Review, Vol.27, No.2, 1989, pp.45-55.