An Optimal Victim Selection Algorithm for Removing Global Deadlocks in Multidatabase Systems

X. Lin & M. E. Orlowska

Department of Computer Science
University of Queensland
Brinsbane, QLD 4072, Australia

Y. Zhang

Math & Computing Department
University of Southern Queensland
Toowoomba, QLD 4350, Australia

Abstract

A time-out mechanism based on a potential conflict graph has been applied to detect global deadlocks. In order to perform the time-out mechanism better, in this paper we propose an optimal victim selection algorithm for resolving global deadlocks in a multidatabase system. The algorithm selects a set of transactions with the minimal abortion cost to resolve global deadlocks. It makes the use of network flow techniques, and runs in time $O(n^3)$, where n is the size of a subset of the global transactions.

1 Introduction

A *multidatabase system* (MDBS) is a federation of independently developed component database systems connected through a communication network. These component database systems are also called *local database systems* (LDBS) in contrast to an MDBS. There are two types of transactions in an MDBS:

- *Local transactions* - that execute at a single LDBS.
- *Global transactions* - that may execute at several LDBSs.

One major issue in transaction management in an MDBS is concurrency control. During the concurrent execution of a set of transactions, a deadlock occurs when they wait, in a circular fashion, for exclusive access to some of the resources held by other transactions in the set. The current developments in concurrency control techniques in multidatabase systems can be classified into two families[2, 4]: one is deadlock free, and another requires deadlock detection. Usually, the concurrency control approach without forcing a deadlock free can potentially provide a greater concurrency degree. But, this should co-operate with an efficient and effective deadlock resolution. Discussions about the advantages and disadvantages of these two approaches are outside the coverage of this paper. The interested reader may refer to [2].

In this paper, we assume that the concurrency control technique used in a multidatabase system may cause the existence of deadlocks. To simplify the discussion, in this paper we adopt the MDBS model presented in [2, 3] which is based on the following assumptions:

1. **At the local level**: no changes can be made to local database systems in order to preserve local autonomy; a local database management system is not able to distinguish between local and global transactions which are active at the LDBS, neither is it able to communicate directly with other local DBMSs to synchronize the execution of a global transaction active at several LDBSs; and each local database management system uses the strict *two-phase locking* protocol [1] for local serializability (that is, local locks are released only after a transaction aborts or commits), and has a mechanism for ensuring freedom from local deadlocks.

2. **At the global level**: the global transaction manager (GTM) has no access to local DBMS; and the GTM submits an operation of a transaction T to an LDBS only if the previous submitted operation of T has been completed.

Thus, a global transaction may wait, at most, at one LDBS each time; and we can assume that each local schedule is serializable and that local deadlocks can be resolved through a local concurrency control approach. However, global deadlocks (that is, the deadlocks among global transactions) may still exist due to either *indirect conflicts* or *direct conflicts* [2].

The time-out mechanism has been introduced [3] to remove the global deadlocks in a multidatabase. Later, an improvement on victim selection is suggested in [4]. In this paper, we present an optimal victim selection algorithm for the implementation of the time-out mechanism to remove global deadlocks. Once it is decided to abort a transaction by an implementation of the time-out mechanism, our algorithm will always choose a set of transactions with the minimum abortion cost, and with better (or the same) abortion effect. The algorithm presented in this paper uses network flow techniques. Particularly, we translate the optimal problem into the maximum flow problem.

The rest of the paper is organized as follows. Section 2 gives a brief overview of the algorithms in [3, 4], our motivations, and an outline of our deadlock resolution. Section 3 shows a translation from the optimal victim selection problem to the maximum flow problem, and then gives the optimal algorithm. Section 4 concludes the paper.

2 Preliminaries

2.1 Related Works

In respect to site autonomy, a GTM has no access to local database management systems. So, indirect conflicts usually may not be precisely determined by a GTM. A *potential conflict graph* has been introduced in [2] to give an approximation status of conflicts, since indirect conflicts cannot be detected by a GTM. A transaction T_i is *active* at site S_j if it has a server at S_j, and if the server is performing an operation of T_i at the site or has completed the current operation of T_i and is ready to receive the next operation of T_i. A transaction that is not active at site S_j is said to be *waiting* at site S_j, provided that it has a server at the site and at least one operation of the transaction was submitted to the site. A potential conflict graph has been described as a directed graph $G = (V, A)$ whose vertex set V consists of the global transactions. An arc $T_j \rightarrow T_i$ is in A if there is a site at which T_j is waiting and T_i is active.

Note that a cycle in a potential conflict graph could be either a real deadlock or a false deadlock due to inaccurate information about conflicts among global transactions; and the potential conflict graph may be changed from time to time. Meanwhile, it is believed that some abortions are more expensive than waiting, and unnecessary abortions result in wasted system resources.

A time-out mechanism BLS has been proposed by Breibart, Litwin and Silberschatz in [3] which cooperates with the potential conflict graph to remove global deadlocks. The algorithm BLS initially issues a time-out to each waiting global transaction, and implements the following two steps once the time-out expires on a waiting transaction T:

BLS1: If there is a cycle in the potential conflict graph at that time, determine the set of all transactions active at the waiting site of T and involved in any cycle through T. If T is older (with respect to timestamp) than all transactions in the set, T continues waiting; otherwise, T aborts.

BLS2: If there is no cycle in the potential graph, T continues waiting.

Two problems of BLS have been reported in [4]: 1) BLS may abort the transactions with expensive cost; 2) BLS may abort a transaction involved in fewer cycles than other transactions in the potential conflict graph. So BLS may abort additional transactions in order to resolve the deadlocks.

A possible improvement PPCG on BLS is outlined in [4], which consists of the following two steps (performed once the time-out expires on T):

PPCG1: If there is at least one cycle in PCG through T, find the set g of the transactions involved in a number of cycles equal to or greater than the number of cycles in which T is involved, and each transaction in g is involved in a cycle through T. If the time-out expiring on T is the last one in g, then abort T. Otherwise choose the least expensive transaction $\tilde{T}$ from g; if the expense of T is the same as that of $\tilde{T}$, then abort T.

PPCG2: Otherwise T continues to wait with a re-initiated time-out.

2.2 Motivations

In our implementation of the algorithm PPCG, we found two problems. One is that if it is decided not to abort T, it is possible that there is a cycle in the potential conflict graph through T which is a real deadlock and may not be broken at that time. In a dynamic environment where new transactions are issued continuously, the potential conflict graph may be extended at the next time-out, and this remaining deadlock may again fail to be broken. Thus, this deadlock may exist forever in the dynamic environment. One may easily construct such a case to illustrate this. Another problem is that the computation of g at PPCG1 involves the computation of the number of cycles in a directed graph. This computation could be expensive (so far, only high order polynomial time algorithms are known).

The main difficulty of PPCG is that it does not take the dynamic environment into account, nor the approximation property about the deadlock information.

Consider that the potential conflict graph only approximately provides the deadlock information. A cycle in a potential conflict graph (PCG) could be either a false or a real deadlock. This results in uncertainty about the information of global deadlocks. The measurement of how many cycles will be broken along with a transaction abortion is not very important, since some cycles could be false deadlocks. Meanwhile when time-out expires on a transaction T, each cycle in the current PCG could be a real deadlock through T. Thus, we should abort all cycles through a transaction T in the potential conflict graph once the time-out expires on T. Clearly, there are two ways to break those cycles through T: one is to abort T, another is to abort a set of other transactions which are through all these cycles. In the following, we propose an alternative way to carry out a time-out mechanism to remove deadlocks in multidatabases. We suggest that once the time-out expires on a transaction T and there is at least one cycle in the PCG through T, instead of aborting T we may find a set of other transactions such that the abortion cost is minimized and the abortion will break all cycles through T.

After aborting a transaction T, all those submitted operations of T should be re-submitted for computation. Obviously, the system resources for aborted operations, which include the communication cost, are wasted. In this paper, we use the number of submitted operations in the execution of transaction T as the *abortion cost*, denoted by $ac(T)$, of T in order to simplify the discussion. The *abortion cost* of a set M of transactions, denoted by $ac(M)$, is the sum of the abortion cost of each transaction in M. We use the following example to illustrate the necessity of our consideration.

Example 1. Suppose that when time-out expires on a transaction T, the execution status of the global transactions and the potential conflict graph are illustrated by Figure 1. The abortion costs are listed as follows: $ac(T) = 8$, $ac(T_1) = 2$, $ac(T_2) = 2$, $ac(T_3) = 2$, $ac(T_4) = 3$, and $ac(T_5) = 2$. Further, suppose that T is not older than either T_1 or T_2. Then according to the algorithm BLS, T will be aborted. Clearly, the abortion cost of $\{T_1, T_2, T_4\}$ is smaller than $ac(T)$, and this abortion will also break all cycles through T. The smallest abortion cost for

breaking all cycles through T is 2 which is required to abort T_3. □

2.3 Our Algorithm OVS

Below, we outline our global deadlock removal algorithm. Once the time-out expires on a transaction T, our algorithm OVS consists of the following three steps:

Step1: Find the "strongly connected component" X from the current PCG, which contains T. If X contains only T, T continues to wait with a re-initiated time-out. Otherwise, go to Step 2.

Step2: Find a subset M of other transactions from X which have the smallest abortion costs such that the abortion of the transactions in M will break all cycles through T. If $ac(T)$ is smaller than $ac(M)$, then abort T. Otherwise abort M, and T either continues to wait with a re-initiated time-out (in case that T still needs to wait for other transactions to finish their operations) or T starts to process the waiting operation.

A *strongly connected component* in a directed graph G is a subgraph $\tilde{G}$ such that:

1. For each pair of vertices u and v in $\tilde{G}$, there are at least two directed paths - one is from u to v and another is from v to u.
2. For each pair of vertices u and v with u in $\tilde{G}$ and v not in $\tilde{G}$, there are no such two directed paths.

Note that any potential conflict graph has no arc that connects the same vertex. Clearly, the potential conflict graph has at least one cycle through T if and only if the strongly connected component X containing T has at least two vertices; and all cycles through T must be in the strongly connected component X. So, at Step 1, that X contains only T means that there is no cycle in the PCG through T. One may find a standard algorithm [6] to find X at Step 1. The algorithm is in linear time with respect to the arc set size of the potential conflict graph. Thus, Step 1 may be implemented in linear time. Section 3 shows that Step 2 can be implemented in $O(n^3)$ by making the use of network flow techniques, where n is the size of the vertex set of the strongly connected component X.

2.4 Networks

In this sub-section, we present some basic knowledge about the network flow problem. An $s-t$ *network* is an *arc weighted directed graph* $N=(V,A,c)$ with two distinguished vertices s and t such that $c: A \rightarrow I$ where I is the positive integer set, and all the arcs attached to s must be the outgoing arcs from s and all the arcs attached to t be the incoming arcs to t. The vertex s is the *source* of N, and t is the *sink* of N. The function c is the *capacity function* of N and its value on an arc a is the *capacity* of a.

A *flow* in an $s-t$ network N is a mapping $f: A \rightarrow I$ such that:

- for each $a \in A$, $0 \leq f(a) \leq c(a)$, and
- for each vertex u other than s and t, $\sum_{a \in A_u^+} f(a) = \sum_{a \in A_u^-} f(a)$, where A_u^+ is the set of the arcs going-out from u, and A_u^- the set of arcs coming to u.

The *maximum flow* problem of an $s-t$ network N is to find a flow f in N such that

$$\sum_{a \text{ attaches to } s} f(a)$$

is maximized.

A *cut* (V_s, V_t) in an $s-t$ network $N=(V,A,c)$ is a partition on V, that is, $V = V_s \cup V_t$, $V_s \cap V_t = \emptyset$, and $t \in V_t$ and $s \in V_s$. The *capacity* of a cut (V_s, V_t), denoted as $c((V_s, V_t))$, is defined as the sum of the capacities of the arcs from V_s to V_t. The *minimum cut* of an $s-t$ network N is a cut (V_s, V_t) such that $c((V_s, V_t))$ is minimized.

We use $V(G)$ to denote the vertex set of a graph G, and $A(G)$ the arc set of G.

3 An Optimal Victim Selection Algorithm

In our algorithm OVS, Step 2 corresponds to solving the following problem.

Minimum Vertex Cut Problem (MVCP)
Instance: Given a vertex weighted directed graph $G=(V,A,ac)$ which is strongly connected and where ac is a mapping from V to the positive integer set I, a vertex $v \in V$.
Question: Is there a subset M of V such that the deletion of M removes all cycles which are through v in G, and $\sum_{u \in M} ac(u)$ is minimized?

Thanks to the developments in the maximum flow problem [6, 5], MVCP can be solved in polynomial time. Below, we translate MVCP to the maximum flow problem.

For a given strongly connected and vertex weighted directed graph $G=(V,A,ac)$ and a given vertex v, we may first modify G into an $s-t$ network G_v, named by the *auxiliary network* of G with respect to v, as follows.

- For each arc a in G, assign the capacity $c(a) = \sum_{u \in V} ac(u) + 1$.
- Split v into two vertices s and t^1. All incoming arcs, in G, to v are moved to attach t^1 with the same capacity as that in G, and all outgoing arcs, in G, from v are moved to attach s with the same capacity. Add one vertex t and an arc $t^1 \rightarrow t$ with the capacity $c(t^1 \rightarrow t) = ac(v)$.
- For each other vertex $u \in V$, split it into two vertices u^1 and u^2. All incoming arcs, in G, to u are moved to attach u^1 with the same capacity as that in G, and all outgoing arcs, in G, from u are moved to attach u^2 with the same capacity. Add one arc $u^1 \rightarrow u^2$ with the capacity $c(u^1 \rightarrow u^2) = ac(u)$.

Based on Example 1, the auxiliary network of the potential conflict graph with respect to T is illustrated in Figure 2.

Obviously, the auxiliary network G_v of G with respect to v has the vertex set size $2|V|+1$, and the

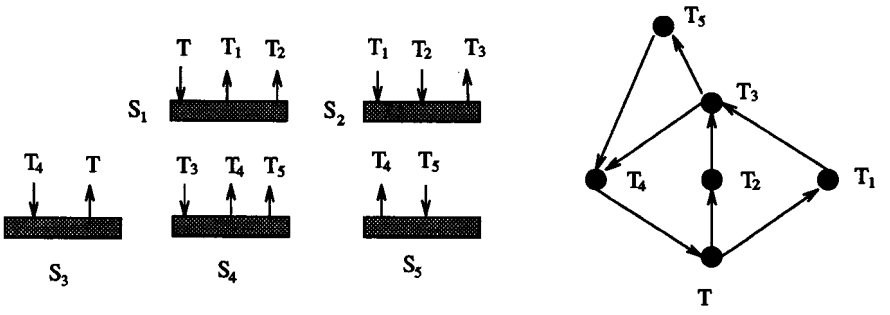

Figure 1: Example 1

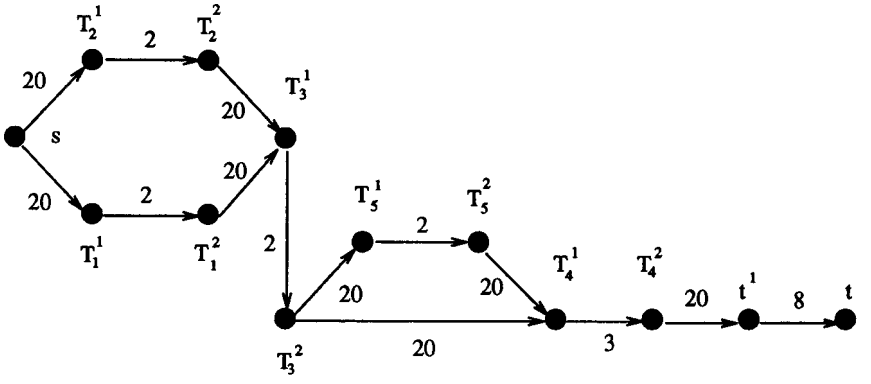

Figure 2: the auxiliary network - an s-t network

arc set size $|A| + |V| + 1$, and G_v is an $s - t$ network. Below, we prove the fundamental Theorem in this paper. Clearly, a vertex set M, which breaks all the cycles through v and has the minimum overall weight, either contains only vertex v or does not contain v.

Theorem 1 *Suppose that a vertex weighted directed graph $G = (V, A, ac)$ is strongly connected, and v is a vertex. Then there is a subset $M \subseteq V$ such that the removal of M breaks all the cycles through v and $\sum_{u \in M} ac(u)$ achieves the minimum value if and only if in the auxiliary network G_v of G with respect v, any minimum cut (V_s, V_t) has the properties:*

- *the capacity $c((V_s, V_t))$ of the cut is equal to $\sum_{u \in M} ac(u)$, and*
- *the set of arcs from V_s to V_t is either $\{u^1 \rightarrow u^2 : u \in M$ where $v \notin M\}$ or $\{t^1 \rightarrow t\}$ in case that M contains only v.*

Proof: We first prove the "if" part by the approach of a reduction to absurdity. Suppose that there is a subset $\tilde{M}$ of V such that the removal of $\tilde{M}$ breaks all the cycles through v and $\sum_{u \in \tilde{M}} ac(u) < \sum_{u \in M} ac(u)$. Without loss of generality, we may assume that $\tilde{M}$ does not contain v. Futher, let $\tilde{A} = \{u^1 \rightarrow u^2 : u \in \tilde{M}\}$. Now we construct $\tilde{V}_s$ and $\tilde{V}_t$ in G_v as follows:

- $\tilde{V}_t = V(G_v) - \tilde{V}_s$, and
- $\tilde{V}_s = \{u^1 : u \in \tilde{M}\} \cup \{s\} \cup V_1$, where V_1 consists of vertices which are on a path, including none arc from $\tilde{A}$, in G_v from s to a vertex in $\{u^1 : u \in \tilde{M}\}$.

From the constructions of G_v, $\tilde{V}_s$ and $\tilde{V}_t$, it immediately follows that for each pair of vertices u^1 and u^2 such that $u \notin \tilde{M}$, either they are all in $\tilde{V}_s$ or none of them is in $\tilde{V}_s$ (because, there is only one arc from u^1 to u^2). Clearly, $t \in \tilde{V}_t$, since the removal of $\tilde{M}$ breaks all the cycles, in G, through v. Also, each u^2 for $u \in \tilde{M}$ is in $\tilde{V}_t$. From the above facts and the constructions of $\tilde{V}_s$ and $\tilde{V}_t$, it follows that $(\tilde{V}_s, \tilde{V}_t)$ is a cut, and the set of the arcs from $\tilde{V}_s$ to $\tilde{V}$ is $\tilde{A}$. Thus $c((\tilde{V}_s, \tilde{V}_t)) = \sum_{u \in \tilde{M}} ac(u)$. It follows that $c((\tilde{V}_s, \tilde{V}_t)) < c((V_s, V_t))$. This contradicts the fact that $c((V_s, V_t))$ is the minimum cut of G_v.

Again, we prove the "only if" part through the approach of a reduction to absurdity. Also without loss of generality, we may assume that M dose not contain v. First, it is clear, from the above proof,

that from M we may construct a cut (V_s, V_t) of G_v such that $c((V_s, V_t)) = \sum_{u \in M} ac(u)$. Suppose that there is a cut (V_s^1, V_t^1) of G_v such that $c((V_t^1, V_s^1)) < c((V_t, V_s))$, and the set π of arcs from V_s^1 to V_t^1 either contains only $t^1 \rightarrow t$ or does not contain $t^1 \rightarrow t$. Without loss of generality, we may assume that π does not contain $t^1 \rightarrow t$.

From constructions of G_v and the cut (V_s, V_t), it follows that π is a subset of $\{u^1 \rightarrow u^2 : u \in V(G)\}$ (noting that the weight of each other arc is larger than $c((V_s, V_t))$). Clearly, the removal of π means that there is no path left from s to t. From these two facts, it can be seen that the removal of M_1 will break all the cycles in G through v, where $M_1 = \{u : \text{ for a } u^1 \rightarrow u^2 \in \pi\}$. From the construction of G_v, it follows that $\sum_{u \in M_1} ac(u) = c((V_t^1, V_s^1))$. Thus, $\sum_{u \in M_1} ac(u) < \sum_{u \in M} ac(u)$. This contradicts the minimum property of M. □

According to Theorem 1, it is clear that the implementation of Step 2 in our algorithm OVS corresponds to finding a minimum cut from an $s - t$ network. The following Lemma from [6] says that we may apply the algorithm for solving the maximum flow problem to finding a minimum cut.

Lemma 1 *For any $s-t$ network $N = (V, A, c)$, the value of the maximum flow equals the capacity of the minimum cut, and a flow f and cut (V_s, V_t) are jointly optimal if and only if*

1. *$f(u \leftarrow v) = 0$ for $u \leftarrow v \in A$ and $u \in V_s$, $v \in V_t$; and*
2. *$f(u \rightarrow v) = c(u \rightarrow v)$ for $u \rightarrow v \in A$ and $u \in V_s$, $v \in V_t$.*

Based on Theorem 1 and Lemma 1, we may carry out Step 2 in the following 4 stages.

s1: With respect to T, obtain the auxiliary network G_T of the strongly connected graph X where each vertex has been assigned a weight corresponding to the abortion cost. Go to s1.

s2: Find a maximum flow f in G_T. Go to s3.

s3: In G_T, let $V_1 = t$ and $V_0 = V(G_T) - V_1$. Then we apply breadth-first search to iteratively extend V_1 and reduce V_0 (that is, we iteratively carry out the following two operations until no changes on V_1 and V_0):

- for an arc $u \rightarrow v \in A(G_T)$, if $u \in V_0$, $v \in V_1$ and $f(u \rightarrow v) < c(u \rightarrow v)$, then remove u from V_0 to V_1; and
- for an arc $v \rightarrow u \in A(G_T)$, if $u \in V_0$, $v \in V_0$ and $f(v \rightarrow u) > 0$, then remove u from V_0 to V_1.

Go to s4.

s4: From Theorem 1 and Lemma 1, it follows that (V_0, V_1) is a minimum cut of G_T, and the set of arcs from V_0 to V_1 is in the form either $\{u^1 \rightarrow u^2 : u \in M$ for some subset M of $V(X)\}$ or $\{t^1 \rightarrow t\}$. (In the later case, let $M = \{T\}$.) Abort the transactions in M.

Because all the cycles through T in the potential conflict graph must be included in the strongly connected component X containing T, by combining this with Theorem 1 and Lemma 1, it follows that

the implementation of the above four steps s1 - s4 can find a subset M of the global transactions such that the abortion of M will break all cycles through T, and $ac(M)$ is minimized.

Clearly, s1 can be implemented in linear time with respect to $|A(X)|$. The most efficient algorithm for solving maximum flow problem takes time $O(|V(G_T)||A(G_T)|log\frac{|V(G_T)|^2}{|A(G_T)|})$ [5], which is slightly better than the algorithm $O(|V(G_T)|^3)$ in [6] for sparse graphs. But the algorithm in [6] is much easier to implement. We suggest applying this algorithm. Thus, s2 can run in time $O(|V(G_T)|^3) = O(|V(X)|^3)$. Note that s3 and s4 can be implemented together, and take $O(|A(G_T)|) = O(|A(X)|)$ (time for breadth-first search). So, we have the following Theorem.

Theorem 2 *Step 2 in the algorithm OVG can be implemented in $O(n^3)$ where n is the number of vertices in X.*

4 Conclusions

In this paper, we provided an alternative implementation of the time-out mechanism to remove global deadlocks in multidatabase systems. The alternative implementation has the guarantee that at each time the abortion cost is minimized, and the abortion effect is either the same as or better (with the possibility of breaking some other cycles not through T) than that in [3]. We transfer the optimization problem to the maximum flow problem, and then we make the use of network flow techniques in our algorithm. In the worst case, the time complexity of our algorithm is cubic.

References

[1] P. Bernstein, V. Hadzilacos and G. Goodman, *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, 1987.

[2] Y. Breibart, A. Silberschatz, and G. Thompson, Reliable Transaction Management in a Multidatabase System, *SIGMOD Record*, 1990.

[3] Y. Breibart, W. Litwin and A. Silberschatz, Deadlock Problems in a Multidatabase Environment, IEEE Data Engineering, January 1991.

[4] O. Bukhres, J. Alm and N. Boudriga, A Priority-Based PCG Algorithm for Global Deadlock Detection and Resolution in Multidatabase Systems, *3rd International Workshop on Interoperability in Multidatabase Systems*, IEEE CS press, 1993.

[5] B. M. E. Moret and H. D. Shapiro, *Algorithms from P to NP, Volume 1: Design and Efficiency*, Benjamin/Cummings, 1990.

[6] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall Publish.