

Deadlock Detection and Resolution in a CODASYL  
Based Data Management System

Philip P. Macri

Bell Laboratories, Inc.  
Piscataway, New Jersey 08854

INTRODUCTION

In a multi-task computer system many different types of situations may occur in which productive computation may be brought to a standstill. One of these is deadlock or "deadly embrace". Some of the earliest investigation into this problem was undertaken by Dijkstra [1], Habermann [2] and Havender [3]. A detailed presentation of the deadlock problem and its ramifications may be found in [1, 2, 3, 4, 5].

Because deadlock is so costly, modern computer system software is designed so that deadlock is either impossible [6, 7] or the probability of its occurrence is minimized at the system level. For example, the INGRES data base management system [7] accomplishes deadlock prevention without compromising data base integrity. Unfortunately, the CODASYL design [8] does not preclude the possibility of deadlock. It leaves the burden of minimizing the probability of its occurrence to the design of the application. There are two parts to the application design: the database design and the software design. Judicious design of both parts can sometimes minimize deadlock. However, deadlock cannot be prevented in all cases. Therefore, deadlock detection and resolution mechanisms are essential for operation in a multi-thread environment. Initially, the CODASYL based data management system employed (DMS 1100 [11]) had very simple deadlock detection and resolution mechanisms. The deadlock detection mechanism was based on the sufficient condition that deadlock exists when the number of transactions registered with the data management system equals the number of transactions locked out of resources. A least number of page alterations criterion was used by the deadlock resolution mechanism to resolve deadlock. That is, the transaction with the least number of altered pages was rolled back in an attempt to resolve deadlock. If this failed to resolve deadlock, the roll back selection process was repeated until deadlock was resolved. It was discovered that these mechanisms seriously degraded throughput on our system and a new approach was needed. This paper contains a description of the deadlock detection algorithm and the deadlock resolution algorithm which was implemented to overcome this deficiency. The former detects deadlock and the latter resolves deadlock in a manner consistent with maximum system throughput.

In order to establish a common framework for discussion, the concepts of lock out and deadlock will be defined before proceeding.

Definition:

A task is said to be locked out of a facility or resource it requires if it is prohibited from acquiring that resource or facility. For example, suppose task A has exclusive use of a file and task B wishes to use the file. Assuming that task B has the authority to use the file in the manner it wishes, task B is locked out of the file by task A. When the file is released by task A, task B may (after possibly competing with other tasks) acquire it.

Definition:

A deadlock is said to exist if every member of a subset of concurrent tasks is locked out of a resource or facility by one or more members of the subset, and each task must have that facility or resource in order to continue processing. One of the simplest deadlock situations occurs when task A has task B locked out of resource 1, while task B has task A locked out of resource 2 and both are in a wait mode for the requested resource to become available.

Since this paper deals only with deadlock in a DMS 1100 data management system, a task is defined to be a transaction which is registered with the data management system. (Transaction registration establishes the necessary communication interface between the task and the data management system.)

DEADLOCK DETECTION AND RESOLUTION

In this section the details of an efficient deadlock detection algorithm are presented. The deadlock detection algorithm uses a directed graph called a lock out network. Lock out networks represent the locking relationships between transactions and are well suited for deadlock detection in DMS 1100. Although it is shown that optimal (with respect to transaction throughput) deadlock resolution is impossible, some considerations concerning heuristic approaches to efficient deadlock resolution are presented. In addition to its role in efficient deadlock detection, the lock out network may be used during deadlock resolution. The topological information contained in the lock out network together with transaction processing costs may be used to select a set of transactions for roll back which is consistent with maximum system throughput. An efficient algorithm for deadlock resolution which uses this information is also presented.

## Deadlock Detection

There are many ways in which allocated resources and lock out in a computer system may be represented by a directed graph [9, 10] in such a manner that the existence of a cycle is a necessary and sufficient condition for deadlock. For convenience, a directed graph representation other than those found in [9, 10] will be used. The type of directed graph used in this paper is called a lock out network. The nodes of the network represent transactions and the directed lines connecting two nodes represent the lock out relationship between the two transactions represented by the nodes. For example, in figure 1 transaction T1 is locked out of resource R1 by transaction T2. Transactions which are not locked out of a resource and are not locking out other transactions would appear as isolated nodes in a lock out network. Because these transactions cannot be involved in deadlock, they are not depicted in the diagrams. Using this model, it can easily be shown that the existence of a cycle in a lock out network is a necessary and sufficient condition for deadlock. Furthermore, those and only those transactions involved in the deadlock are represented as nodes in the cycle corresponding to the deadlock in question. For example, figure 1 illustrates a simple deadlock situation in which two transactions are deadlocked. Transaction T1 is locked out of resource R1 by transaction T2 which in turn is locked out of resource R2 by transaction T1. Figure 2 illustrates a more complicated deadlock situation involving two deadlocks.

The deadlock detection algorithm is performed by the blocked transaction whenever a transaction is locked out of a resource for a reason which could produce deadlock. Deadlock is detected by a path search which finds all simple cycles containing the starting point of the search. The node representing the blocked transaction executing the detection algorithm serves as the starting point of the search. Hence, all deadlocks are detected by finding all simple cycles containing the starting nodes. This frequency of deadlock detection was chosen over periodic execution for the following reasons:

1. It reduces the complexity of the resolution algorithm.
2. Deadlock is resolved as soon as it occurs. Thus the processor is kept busy.
3. Periodic execution offers little, if any, overhead advantage.

## Deadlock Resolution

Once deadlock is detected it can be resolved by eliminating the cycle causing the deadlock. Breaking the cycle causing deadlock is one method of accomplishing deadlock resolution. This can be done by either total or partial transaction roll back. In total roll back all of a transaction's computation is undone. Whereas, in partial roll back, only the computation from the present back to a "roll back point" is undone. In figure 1, deadlock resolution can be accomplished through partial roll back by backing up either transaction T1 to a point just prior to its locking of resource R2, or by backing up transaction T2 to a point just prior to its locking of resource R1. In order to undo the least amount of processing by partial roll back, the entire state of a transaction must be saved everytime a transaction applies a lock (eg. alter a page) which could produce deadlock. This is a considerable amount of processing, because locks which could produce deadlock are frequently applied. However, "roll back points" which are not as "fine" as the above could be established. For example, a roll back point could be established at every tenth page alteration. Saving the state of the transaction in DMS 1100 would mean saving a copy of the entire core resident application program, all registers, etc. The volume of data necessary to accomplish even a crude partial roll back in the current implementation is too enormous to make this approach practical. Therefore, total roll-back (all of the transaction's processing is undone) was chosen to resolve deadlock.

Even if the state of a transaction could be captured in an optimal manner, there are other reasons why partial roll back is not an attractive approach to deadlock resolution. Depending on the coarseness of partial roll back and the frequency of deadlock, partial roll back, in terms of system throughput, could be more costly than total roll back. Because the amount of deadlock in our application is not expected to be high, the additional overhead necessary for partial roll back would not represent any savings over total roll back.

Furthermore, deadlock must be resolved on the basis of some criterion. As discussed below, a minimum processing cost criterion was selected. The additional overhead necessary to resolve deadlock on this basis may outweigh any benefits obtained from partial roll back. This additional deadlock resolution overhead requirement for partial roll back consists of two parts: the additional processing costs in the execution of the deadlock resolution algorithm and the maintenance of the information necessary for deadlock resolution. For each deadlocked transaction, the resolution algorithm must be able to determine the partial roll back point. Furthermore, in order to resolve deadlock on a processing cost basis, the resolution algorithm must also have access to or be able to compute the amount of processing that would be wasted if the transaction were rolled back to the partial roll back point. There are tradeoffs between the amount of processing each of the two parts must perform. If the necessary information is kept on a sequential file with the roll back information, then the maintenance of this information would not be costly. However, the deadlock resolution algorithm alone would perform as much or more work than the total roll back procedure itself. On the other hand, if this information is indexed then the resolution algorithm would incur little additional processing. The overhead in the maintenance of this information depends upon the coarseness of partial roll back. However, except for very coarse partial roll backs, this overhead could outweigh the

partial roll back benefits because the incidence of deadlock is expected to be low, and the frequency of applied locks is high.

However, even with total roll back, optimal (in the sense of system throughput) deadlock resolution requires not only the knowledge of all resources required, but the sequence in which they are needed. The following example illustrates this point. Suppose that in figure 3a deadlock is resolved by rolling back transaction T3 allowing transaction T2, T4 and T5 to resume processing. While processing, transaction T2, before releasing resource R1, required resource R5. However, resource R5 is held by transaction T1 in such a manner as to lock out transaction T2. This resulted in the deadlock illustrated in figure 3b. However, if in figure 3a, T2 is rolled back instead of T3, the nature and sequence of resource requirements is such that the remaining transactions would finish processing without the occurrence of another deadlock. (Because resource needs are sometimes dependent on information contained in the database, knowledge of resource requirements may necessitate transaction processing.) However, if this knowledge of resources is available, there would be no need for deadlock detection and resolution. Deadlock could be prevented by proper active sequencing of transactions and their acquisition and releasing of resources.

Once deadlock is detected circumstances may exist under which deferral of deadlock resolution may improve throughput. Suppose the processor is highly utilized at the moment when the deadlock illustrated in figure 4 is detected. Selecting a transaction for roll back at this time may not be beneficial to throughput because the processor is busy with other transactions. Furthermore, as illustrated in figure 4, the releasing of resources of the transactions selected for roll back may not allow any of the other transactions in the cycle to resume processing because they may be locked out by other transactions. However, without the resource requirement knowledge described above, it is possible that resolution deferral may result in lost throughput. Because of its complexity and its limited advantages, resolution deferral was not implemented.

The above discussion gave some topological properties of lock out networks which may be used in deadlock resolution. However, there are other factors which may be useful in deadlock resolution. In our environment, transaction roll back selection should be made in a manner consistent with maximum system throughput. Because it requires complete prior knowledge of resources required by all processing transaction, optimal deadlock resolution is impractical. Therefore, some heuristic device must be used in roll back selection. One such heuristic device is to minimize the amount of computer processing wasted because of deadlock resolution. Using this criterion, the set of transactions which represents the least amount of computer processing consumed (consequently wasted) and still resolve all deadlocks will be selected for roll back. Because rolled back transactions must be reprocessed, the amount of wasted processing and consequently the cost to system throughput caused by rolling back a transaction may be decomposed into two additive components: The processing cost to get the transaction to its current state and the processing cost to roll back the transaction. The processing cost to bring the transaction to its current state may be estimated by using computer accounting information. The number of altered pages may be used to calculate the processing costs needed for roll back. In our system, both parameters are readily available.

Processing costs may be combined with topological properties to determine which transactions are to be rolled back. This was done using the following algorithm:

1. If one deadlock cycle is detected, the transaction to be rolled back is the one in the deadlock cycle with the smallest amount of wasted work.
2. If two deadlock cycles are detected, a compound deadlock situation exists. In this case, three sets are defined. Set A is the set of all transaction common to both cycles. Set B is the set of transactions in the first cycle but not the second and set C is the set of all transactions in the second cycle but not in the first. The transaction with the minimum amount of wasted work is found for each set. The wasted work for the selected transaction from set A is compared against that of the transaction from set B plus that of the transaction from set C. If that of set A is smaller, the transaction from set A is rolled back. If that of set B plus set C is smaller, the transaction from set B and the transaction from set C are both rolled back. If either set B or set C is null, a transaction from set A must be rolled back in order to resolve deadlock. In this case the transaction from set A is rolled back. For example, in figure 2, set A = {T1,T2}, set B = {T4} and set C = {T3,T5}.
3. If three cycles have been found, the transaction in the subset of transaction involved in all three deadlocks that has the minimum wasted work will be selected for roll back. This strategy has been selected to reduce computational complexity. Also, the probability is very low that a three cycle deadlock will occur, and if it does, the probability that two or three transactions would be selected for roll back rather than one is even lower.
4. The transaction detecting deadlock must be involved in the detected deadlock. This is due to the fact that deadlock detection is performed whenever a transaction is locked out of a resource for a reason which could produce deadlock. The transaction itself executes the deadlock detection code. This makes it possible for the final part of the resolution algorithm. If four or more deadlock cycles are found, the transaction detecting the deadlock is rolled back. This choice was made for computational efficiency and because of the very low probability of any other transactions being rolled back in such a situation, were it ever to occur.

## RESULTS

A much simpler deadlock detection and resolution mechanism was in use prior to the implementation of mechanisms described in this paper. That deadlock detection algorithm was based on the fact that deadlock exists when the number of transactions locked out of resources equals the number of transactions registered with the data management system. Unfortunately, this is only a sufficient condition which may not occur even though deadlock is present. Sufficiency was forced to occur by prohibiting transactions from registering with the data management system when the number of transactions locked out of resources was greater than one. If deadlock exists, sufficiency would have eventually occurred because the number of transactions registered with the data management system decreased as transactions de-registered and the number of transactions locked out of resources increased. Eventually either the number of blocked transactions decreased to one or deadlock occurred. In the former, the policy of prohibiting transaction registration was relaxed and in the latter, deadlock resolution began.

Deadlock was resolved by rolling back transactions, one at a time, until at least one transaction was not blocked. The least number of data page alterations criterion was used to select a transaction for roll back. Unfortunately, as was often the case, the transaction selected for roll back was not involved in deadlock. Thus, processing was needlessly wasted by the blind roll back selection process. However, the prohibition of transaction registration was the most damaging aspect of this mechanism. It severely penalized system throughput. The mechanism described in this paper approximately doubled benchmark transaction throughput over the simpler mechanisms.

The deadlock resolution algorithm described in section II was compared to three other resolution algorithms. In terms of wasted processing the algorithm was:

- A. 40% better than using a number of altered data pages criterion. In this comparison, the number of altered pages was used in place of wasted processing in the algorithm described in section 2. The poorer performance was attributed to the fact that the number of altered pages is a measure of the amount of processing necessary for roll back. It is not a good estimate of the amount of processing that must be repeated.
- B. 43% better than the simpler deadlock detection and resolution mechanism described above. The somewhat poorer performance as compared to A is attributed to blind roll back. That is, transactions not involved in deadlock may be rolled back.
- C. 74% better than rolling back the transaction detecting deadlock. Deadlock detection is performed by a transaction when it is locked out of a resource for a reason which could produce deadlock. Therefore, if deadlock exists, this transaction must be in all deadlock cycles. Rolling back this transaction will resolve deadlock.

## REFERENCES

- [1] Dijkstra, E. W., "Cooperating sequential processes," in Programming Languages (F. Genuys, ed.) Academic Press, N. Y. 1968, (43-112).
- [2] Habermann, N., "Prevention of system deadlock" Comm ACM 12, 7 (July 1969), 373-377.
- [3] Havender, J. W., "Avoiding deadlock in multi tasking systems", IBM Sys. J. 7,1 (1968), 74-87.
- [4] Shoshiani, A. and E. G. Coffman, Jr., "Sequencing tasks in multi-process, multiple resource systems to avoid deadlocks". Proc. 11th Symp. Annual Switching and Automatic Theory (Oct 1970), 225-233.
- [5] Coffman, E. G., Jr. and P. J. Denning, Operating Systems Prentice-Hall, Inc., Englewood Cliffs, New Jersey.
- [6] Chamberlain, D. et al, "A Deadlock-Free Scheme for Resource Locking in a Data-Base Environment", IBM Research Laboratory, San Jose, Cal. March 1974.
- [7] Stonebraker, M. "High Level Integrity Assurance in Relational Data Base Management Systems" Memorandum No. ERL-M473 August 16, 1974, Electronics Research Laboratories, College of Engineering, University of California, Berkely.
- [8] "CODASYL Data Description Language", Journal of Development, June 1973, U. S. Department of Commerce, National Printing Office, Washington, D. C. 1974.
- [9] Holt, R. C. "On Deadlock in Computer Systems", Technical Report CSRG-6; Computer Systems Research Group; University of Toronto; January, 1971.
- [10] King, P. F. and A. J. Collmeyer "Database sharing - An efficient mechanism for supporting concurrent processes" pgs 271, 275, Proceedings of the National Computer Conference, 1973, Chicago.
- [11] UNIVAC 1100 Series, Data Management System (DMS 1100) Data Manipulation Language Programmer Reference, UP-7908.

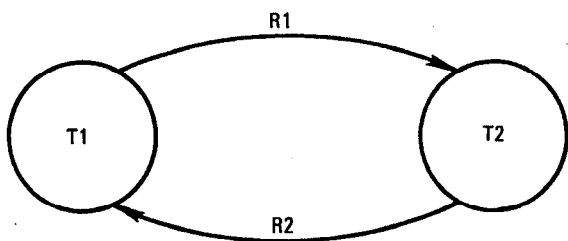


FIGURE 1

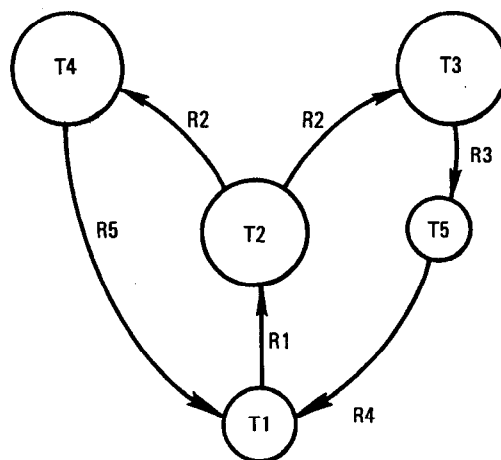


FIGURE 2

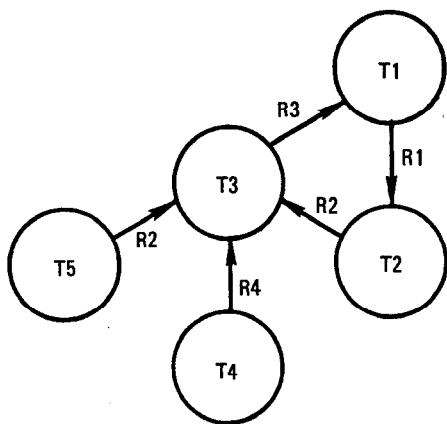


FIGURE 3a

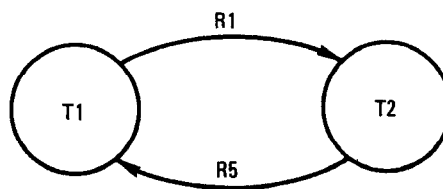


FIGURE 3b

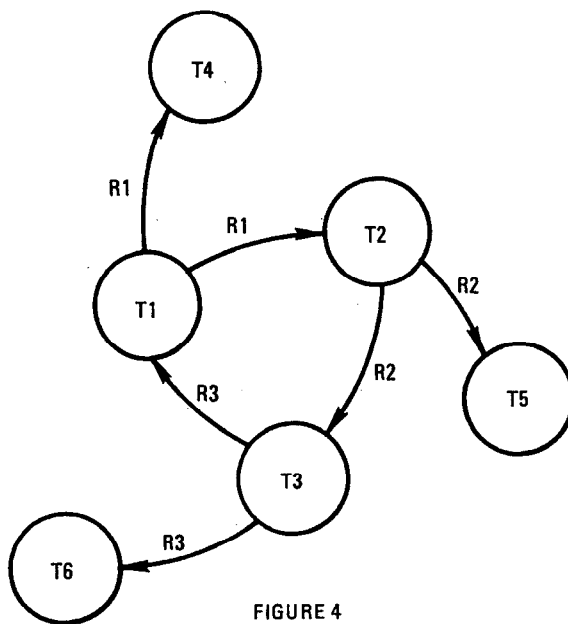


FIGURE 4