TECHNICAL PAPERS

DETECTION OF DEADLOCKS IN DISTRIBUTED DATABASE SYSTEMS*†

T. ANTHONY MARSLAND AND SREEKAANTH S. ISLOOR

Department of Computing Science, University of Alberta

ABSTRACT

Designers of DBMS have faced many problems with concurrent access and update of information. When a database is distributed over several dissimilar computer systems, such problems as detection and avoidance of process deadlock are compounded. Additional communications may be necessary between the computers to determine the relationships between the data resources and processes.

In this paper a thorough discussion of the drawbacks of, and the problems involved in, previous proposals to handle deadlocks in distributed DBMS is provided. Earlier algorithms have required substantial communication and coordination between computers should a deadlock arise, thus delaying corrective action. The concept of "on-line" deadlock detection is introduced here; using a graph-theoretic model to represent the process interactions, a new algorithm for the immediate detection of deadlocks in distributed databases is proposed. The theorems upon which our algorithms are based are presented without formal proof. A tutorial approach is taken and application of the theorems is illustrated through a set of examples. Several highlights of our proposals are emphasized.

RÉSUMÉ

Les dessinateurs du DBMS (système de gestion de banques de données) ont rencontré plusieurs problèmes avec la modification et l'accès concurrents de l'information. Quand une banque de données dessert plusieurs systèmes d'ordinateurs dissimilaires, des problèmes tels que la détection et l'évitement des situations de "cadenas" dans le processus sont multipliés. Des communications supplémentaires entre les ordinateurs peuvent devinir nécessaires afin de déterminer le rapport entre les ressources des données et leur manipulation.

Cet écrit discute minutieusement les inconvénients et les problèmes impliqués par des propositions préalables sur la manipulation des situations de "cadenas" dans les DBMS distribués. La situation de "cadenas" écheánt, les algorithmes antérieurs nécéssitaient une communication et une coordination substantielle entre les ordinateurs, ce qui retardait une action corrective. Le concept de la détection directe ("on-line") des situations de "cadenas" est présenté; utilisant un modèle basé sur les graphes pour représenter les intéractions des processus, nous proposons un nouvel algorithme pour découvrir immédiatement les situations de "cadenas" chez les banques de données distribuées. Les théoremès sur lesquels sont basés nos algorithmes sont présentés sans preuve formel. Nous avons pris une approche didactique et nous avons illustré les applications des théoremès par des exemples.

*This research is supported in part by a grant from the Canadian Natural Sciences and Engineering Research Council.

†Received 2 November 1978; revised 27 March 1979 and 22 June 1979.

1 Introduction

With the increases in database size, complexity, diversity of use, and users' strong preference for interactive computer systems, the necessity for additional computing resources grows rapidly. Replacement by higher performance components is an expensive way of growing. Distributing several system functions over a network of computers has been projected as an economic panacea to the expansion problem which will provide improved performance, as well as enhanced accessibility of resources.(1,2) A database management system is said to be "distributed" if hardware or processing logic, data, the processing actions, and the operating system components are dispersed on multiple computers which are logically and physically interconnected. In such a system, data may be replicated at several sites or on separate storage devices; the processing logic co-operates and interacts through a communication network under decentralized system-wide control.

Major advantages offered by a distributed database over a centralized system are as follows.

(a) Reliability: With data redundantly stored on multiple computers, the system is not susceptible to total failure when a single computer component breaks down.
(b) Responsiveness: The close proximity of the data enhances accessibility of resources and improves system performance.
(c) Expansion: The system lends itself to incremental upward scaling.

From the viewpoint of system implementation, the degree of distribution of system functions is indicated by the amount of decentralization in hardware, control point, and database organization. Their overall operations, although autonomous, are characterized by cooperative interactions. A taxonomy of distributed processing systems characteristics has been provided by Chang,(3) Eckhouse et al.,(4) Lientz and Weiss,(5) and Marsland and Sutphen.(6)

Distributed data management is a product of the "marriage" between database management systems (DBMS) and computer networks. As emphasized earlier such information networks offer many potential advantages, but the realization of generalized systems is slowed by formidable obstacles. As pointed out by Fry and Sibley,(7) dispersed information systems share numerous design problems with both DBMS and computer networks. In addition, they produce several other difficulties, such as that of locating and updating redundant data. Peebles and Manning(8) and Maryanski(9) have listed major problems in the area of distributed databases.

The designers of distributed databases are also haunted by conventional problems, such as deadlock, data integrity, and security, which are intensified in a network environment. Deadlock is an unfortunate side effect that arises because of the necessity for processes to establish exclusive control over data resources while modifying them. When two or more processes request exclusive access to data resources held by each other, they reach a state in which each one perpetually blocks the other(s) from execution. The probability of deadlock in large integrated databases (with a high degree of concurrent access and update) grows with increased usage, making a re-examination of the problem important.

Considerable research has been done on this problem in operating systems. It has been shown that, even though a resource is allocated to a process, it may be possible to suspend the process and pre-empt the resource, while preserving the current state of the process and that resource for a later resumption. For instance, if a CPU is the resource held by a process, then the information that must be preserved in case of preemption consists typically of the contents of the CPU registers and the current "program status word." Briefly, three broad categories of algorithms have been proposed:(10) (i) detection, (ii) avoidance, and (iii) prevention.

In DBMS the problems are somewhat different. Typically, preemption of a data resource entails abortion, rollback, and restart of one or more processes, and must be done in such a way as to maintain the correctness and consistency of the database. This is generally expensive and significantly complicates the problem. A score of research papers have reported on different aspects of the deadlock problem in DBMS, most notable being the works of Shoshani and Bernstein,(11) King and Collmeyer,(12) Chamberlin et al.,(13) Schlageter,(14) Stearns et al.,(15) and Eswaran et al.(16) For all practical purposes these researchers have restricted themselves to centralized DBMS. Relatively little work has been reported in distributed systems.

The fact that all information needed to detect deadlocks in geographically distributed databases is not necessarily available at any single installation makes their timely detection difficult. Communication delays may further complicate the problem of obtaining an accurate view of the status of the computer network. Rollback and recovery in distributed DBMS also become very involved and may incur a heavy communication overhead. Our detection algorithm reduces the difficulties associated with these communication delays by ensuring that each site can determine independently, and at the time of allocation, whether or not a particular resource allocation will lead to a deadlock. An "on-line" detection scheme may be defined as one which invokes this mechanism for every allocation request. Isloor and Marsland(17) provide a discussion of different approaches to the problem, and their relative advantages/disadvantages. To maintain the operational fidelity of any distributed system with respect to the problem of deadlock, it has been argued that detection techniques may be more advantageous than avoidance or prevention methods.(17) Supporting arguments are provided by Peebles and Manning(8) and Le Lann,(25) as paraphrased in Section 5.

In Section 2 we identify the shortcomings of some alternative proposals to detect and prevent deadlocks in distributed DBMS. A graph-theoretic model is presented in Section 3 to represent process interactions and is used to establish necessary and sufficient conditions for the existence of deadlocks. A network configuration is shown which is used as a running example. Section 4 provides an intuitive insight into our approach to detection in distributed DBMS. In Section 5 the idea of on-line deadlock detection is introduced, and a detection technique is suggested. Finally the two remaining sections summarize several highlights of our proposal and indicate further research possibilities.

2 Survey of Previous Approaches

Prevention of deadlocks in distributed DBMS has been the subject of papers by Chu and Ohlmacher(18) and Maryanski.(19) In their first approach Chu and Ohlmacher require that all data resources be allocated to the processes before initiation, which may thus be needlessly delayed. Their second technique is based on the concept of a process set, which is a collection of processes with access to common data resources. A process is allowed to proceed only if all data resources required by the process and the members of its process set are available. In Maryanski's proposal each process has to communicate its shared data resources list (conceptually similar to a process set) to all other processes before it can proceed. This shared data resources list is determined by using what is called a process profile, which contains a list of data resources that can be updated by the process. The communication and computation of process sets (or the shared data resources list), which are performed continually as processes enter and leave the system, make heavy demands on the system.

Chandra et al.,(20) Mahmoud and Riordon,(21,22) and Goldman(23) have proposed techniques for deadlock detection in a network environment. At each installation, Chandra et al. require the maintenance of a resource table which contains information pertaining to local resources allocated to processes, processes waiting for access to local resources, remote resources allocated to local processes, and local processes waiting for access to remote resources. The type of access requested by the process is also stored. They hypothesize that by using such tables, well-known algorithms for detecting deadlocks in a single system can be extended to detect them in the computer network by communication between installations and appropriate expansion of resource tables. Schemes to expand resource tables in a network environment are included. Goldman, however, has shown an example in which a deadlock is not detected by their proposed scheme.

The Centralized Control approach to deadlock detection in distributed DBMS[21,22] creates an overall picture of the global network status by utilizing file queues and pre-test queues (a queue of requests which can only be granted at a future time) received from all installations in the network. As the identifiable unit of object-data becomes smaller in size, message congestion at the control node increases to degrade the network performance. The authors[21,22] also propose a Distributed Control approach in which, in a network of n computers, each installation transmits $(n - 1)$ identical messages containing status and queues of files. Each installation thus receives $(n - 1)$ different messages. As shown by Goldman this approach also has a flaw in which a certain deadlock goes undetected. In any case, all proposals[20–23] require the communication of large tables between installations.

The detection schemes of Goldman[23] are based on the creation and expansion of an Ordered Blocked Process List (OBPL). An OBPL is a list of processes each member of which (except the last) is waiting for a data resource held by the next process in the list. Whenever an OBPL is transmitted between installations, a data resource name is inserted into the "single data resource identification part" of the OBPL. The last process in the list either has access to, or is waiting for, that resource. In the former case the state (blocked or active) of the last process in the OBPL must be determined, while in the latter case one needs to know the state of the process which *holds* the data resource. Goldman proposes techniques to identify these states and to ascertain the inevitability of deadlock.

Even though Goldman's method is sound, it does have some shortcomings. For instance, no process may have more than one outstanding resource request, which is not generally the case in real world situations as illustrated in the Appendix to this paper. Also, when several readers share access to a data resource, Goldman requires that each reader make a copy of the original OBPL for its own use (since if one of the readers is deadlocked, then so is any process which requests access to its shared resource). It is possible that OBPL s, while undergoing expansion, could be transferred (sequentially) among several installations (or several times between the same two installations) before a deadlock is detected. Furthermore, OBPL s could become large, leading to substantial overhead, especially when records or entities are considered as data resources instead of files.

The primary disadvantage of all the existing methods is that they cannot recognize that deadlock is imminent without substantial communications between the other computers in the network. Thus, the algorithms described above cannot be used effectively for on-line detection since they are susceptible to "synchronization error," in which either a deadlock is indicated where one no longer exists or a deadlock occurs and is not recognized – when two autonomous computers concurrently allocate resources before advising each other of their actions.

3 Preliminary Notions and Conditions for Deadlock

Certain important notions with direct relevance to our techniques are introduced and illustrated in this section with a running example, which is used to provide an intuitive insight into our approach. For several other aspects of deadlock the reader is referred to the papers by Coffman et al.[24] and Holt.[10] We choose Holt's graph-theoretic deadlock model, proposed for operating systems, and extend it to represent process interactions in a distributed database.

The set of data resources (typically files, fields, records, or entities), represented by $D = \{D_1, D_2, ..., D_m\}$, is held by a set of processes denoted by $P = \{P_1, P_2, ..., P_n\}$, running concurrently in a network of computers. A directed graph with nodes corresponding to either a process or a data resource in these sets, and with edges between nodes representing process interactions in the system, can be used to depict system status.

We state this formally, as follows: A *system graph* $G_s = (N, E)$ is a directed bipartite graph* whose disjoint sets of nodes are those corresponding to P and D, respectively called *process nodes* and *data resource nodes*, such that $N = P \cup D$. An edge from a data resource D_i to a process node P_j, denoted (D_i, P_j), is called a *resource-process access* (RPA) edge which specifies that the data resource D_i is held by process P_j. Similarly, a directed edge from a process node P_r to a data resource node D_s, denoted (P_r, D_s), is called a *process-resource wait* (PRW) edge, and indicates that the process P_r is waiting for access to the data resource D_s. E is the union of the sets of RPA edges and PRW edges. The type of access granted to an RPA edge, or requested by a PRW edge, is indicated by the letter e or s for exclusive or shared access.

Remarks

A process cannot hold a data resource and be simultaneously waiting for access to it (i.e., cannot be self-blocking). It is, thus, necessary for the

*For the definitions and understanding of graph-theoretic terms, see: N. Deo, *Graph theory with applications to engineering and computer science.* Englewood Cliffs, NJ: Prentice Hall, Inc., 1974.

process to declare its most restrictive use of a data resource, in order to prevent the process from getting blocked waiting for exclusive access when it already has shared access.

The *reachable set* of a node N_j of the system graph G_s, denoted by $R(N_j)$, is the set of all nodes in G_s such that there exists a path directed from N_j to all nodes in $R(N_j)$. The notion of a reachable set, first introduced by Holt,(10) facilitates the design of algorithms which do not need the transmission of large tables in order to detect deadlocks. Reachable sets can also be used in on-line deadlock detection, as proposed in this paper.

A deadlock situation may arise when the following necessary conditions hold: (a) the processes request exclusive control of data resources (for updating); (b) the processes are holding data resources allocated to them, and are waiting for additional ones (all acquired data resources should be held until process completion for consistency reasons(16)); and (c) the pre-emption of data resources from processes is not permitted.

(Pre-emption is the reclaiming of a resource by the system, and requires the support of a rollback and recovery mechanism.) Characteristically, a state of *deadlock in a "circular wait"* condition exists when, in a circular chain of processes, each process holds one or more data resources and has requested access to at least one data resource held by the next process in the chain.

Example 3.1

In figure 1, $\{P_0, P_1, P_2\}$, $\{P_3, P_4\}$, $\{P_5, P_6\}$, $\{P_7, P_8\}$ are subsets of processes that exist in computers C_1, C_2, C_3, and C_4 respectively. Data resources $\{D_0, D_1, D_2\}$, $\{D_3, D_4, D_5\}$, $\{D_6, D_7\}$, $\{D_8, D_9, D_{10}\}$ reside at C_1, C_2, C_3, C_4. The RPA edges are shown by solid lines, the PRW edges by dashed lines. The closely dotted and the sparsely dotted lines represent RPA and PRW edges that have not yet been introduced into the system. These future requests are included to illustrate the various aspects involved in on-line detection. At computer C_2 of the network, P_3 holds D_3, D_4, and D_6, and is active. P_4 holds D_5 and is waiting for access to D_4 and D_2, and hence is blocked. Processes P_1, P_4, P_5, P_6, and P_7 are blocked, and P_2, P_3, and P_8 are active. The reachable sets of nodes D_6, P_4, and P_2, for instance, are $\{P_3\}$, $\{D_4, P_3, D_2, P_2\}$, and $\{\phi\}$ respectively. Since P_6 holds D_7 for shared access, a request by P_5 for shared access to D_7 (which results in the introduction of the RPA edge (D_7, P_5)) may cause P_5 and P_7 to be deadlocked.

3.1 *Necessary and sufficient conditions for deadlock*

A process is blocked if it waits for a data resource held by another process. This implies the existence of a path in the system graph from this process

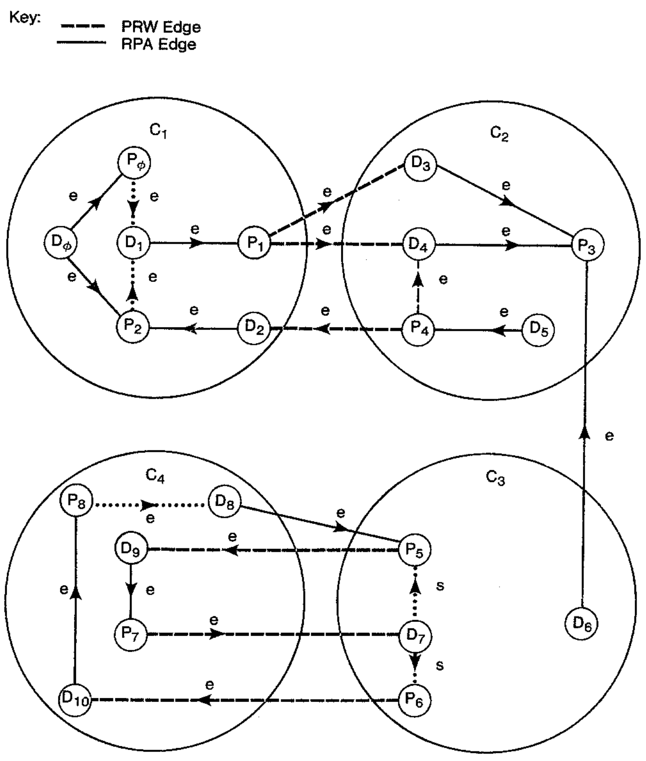

FIG. 1. A system graph for a network configuration with four computers, C_1, C_2, C_3, and C_4, a set of concurrent processes $\{P_i\}_{0\leqslant i\leqslant 8}$ and a set of data resources $\{D_j\}_{0\leqslant j\leqslant 10}$.

to the other. In a circular chain of waiting processes, it is possible to reach the starting process node by traversing the system graph which in turn means that the process node belongs to its own reachable set. Conversely, if a process node belongs to its own reachable set, the process is involved in a circular blockage. This is a necessary and sufficient condition for the existence of deadlock. For instance, in figure 1, the introduction of the PRW edge (P_8, D_8), consequent to P_8 requesting access to D_8, forms a circular chain of waiting processes P_5, P_6, P_7, and P_8. It is evident that there is a directed path starting and terminating at P_8 traversing D_8, P_5, D_9, P_7, D_7, P_6, and D_{10} (in that order). In other words, all the process nodes P_5, P_6, P_7, and P_8 share the same reachable set. We state these necessary and sufficient conditions in Theorem 1 and Corollary 1, the formal treatment and proofs of which appear in reference (17).

Theorem 1
A process is deadlocked in a circular wait condition if and only if the reachable set of the corresponding process node contains the node itself. □

Corollary 1
A set of processes is deadlocked in a circular wait condition if and only if each process node belongs to its own reachable set and the reachable sets of all nodes are the same. □

It is possible to find a process blocked forever, but not deadlocked, if it is waiting for a process which is involved in a deadlock. For example, in the system graph of figure 1, introduction of the PRW edge (P_3, D_5), consequent to P_3 requesting access to D_5, causes P_3 and P_4 to be deadlocked. Thus P_1 which waits for P_3 to release D_3 and D_4, is blocked forever. It can be shown that the set $R(P_1) = \{D_2, D_3, D_4, D_5, P_2, P_3, P_4\}$ contains the sets $R(P_3)$ and $R(P_4)$. We state a sufficient condition for such a situation in Corollary 2, which is treated formally elsewhere.(17)

Corollary 2
A process not necessarily deadlocked in a circular wait condition is blocked forever if its reachable set contains that of a process which is deadlocked. □

4 Deadlocked Detection in Distributed DBMS

For the algorithms proposed in this paper, all the necessary information to detect deadlocks is made available through a system graph at each installation. Maintaining the system graph is trivial, and requires communication only for processes and resources that are global in nature. For processes that are local, accessing only those local resources which have no global interactions of any kind, communication is not necessary. It is believed that for transaction processing systems over 95–99% of processes fall into this category.* However, to maintain a system graph for global processes, or for those that interact with one, communication is required. This communication provides little impact on the network, unlike the earlier schemes which make very heavy demands in order to determine the true network status. Processes which are local at a par-

*See P.A. Bernstein et al., "The concurrency control mechanism of SDD-1: a system for distributed databases (the fully redundant case)," IEEE *Trans. on Software Engg.*, *SE-4*(3), May 1978, pp. 154–168, and M.R. Stonebraker, "Concurrency control and consistency of multiple copies of data in distributed INGRES," IEEE *Trans. on Software Engg.*, *SE-5*(3), May 1979, pp. 188–194.

ticular instant could become global at a later time, necessitating the transmission of a collection of accumulated resource allocations to the various installations. Transition of a local process to global status leads to a small incremental change in the size of the global system graph. The ensuing communication is still modest. Communication activity in our approach is modest in the sense that it is incremental and is dispersed over a period of time. Other approaches rely on simultaneous exchange of status for each site in the network. The transmission delays due to huge message traffic can lead to synchronization problems, which our more responsive method minimizes. Our approach also avoids the message congestion caused by simultaneous transfer of large tables from every installation. The advantage of this method lies in its utility for on-line deadlock detection in distributed DBMS, which in turn means the corrective action can be taken earlier.

Given the system graph G_s, the two significant steps in the detection of deadlocks are: (i) to determine the reachable sets of all the nodes (maintained by continual and incremental updating); and (ii) to find out if the necessary and sufficient conditions for the existence of a deadlock are fulfilled, by utilizing the reachable sets. These steps can be illustrated by an example. Assuming that P_8 requests access to D_8 in the configuration of figure 1, the PRW edge (P_8, D_8) is introduced. The inclusion of this edge causes P_8, P_5, P_7, and P_6 to be deadlocked. To detect such a deadlock, our mechanism determines reachable sets of all nodes in G_s. In the example considered, the reachable sets of P_5 through P_8 are the same and equal $\{D_8, P_5, D_9, P_7, D_7, P_6, D_{10}, P_8\}$. The existence of deadlock is detected by noting that each process in $\{P_5, P_6, P_7, P_8\}$ belongs to its own reachable set.

5 "On-Line" Detection of Deadlocks in Distributed DBMS

When dealing with concurrent database accesses, little is known about the probability of interference or deadlock. For transaction processing systems, Peebles and Manning[8] firmly believe that interference is rare, and that *elaborate avoidance or prevention mechanisms would not be economical.* We agree and advocate the use of deadlock detection in distributed systems. Further, to quote Le Lann,[25] "our conclusion will be that for systems which include a partitioned database and which provide for storage of pending requests, maintenance of internal integrity boils down to a problem of deadlock avoidance or detection with distributed control." As a consequence, and in view of the present-day trend towards increased concurrent access in systems, we recommend the use of on-line deadlock detection in distributed systems. It is our belief that such a method contributes substantially to increasing concurrency.

In our view, deadlock prevention schemes are not justifiable for use in distributed systems. Processes that are not known to be nonconflicting need extensive coordination and, in general, substantial communication among installations is necessary before process initiation. This affects system performance by lowering the degree of concurrency. The past use of prevention principles was acceptable because of low levels of concurrency in systems rather than any inherent superiority.

In on-line detection, every installation can determine whether or not allocating one of its data resources to a process residing on another computer will lead to deadlock. This is facilitated by the ready availability at each installation of the system graph and the reachable sets, which are continually updated as edges are added or deleted. The data resource allocation decision is transmitted by the access controller at the installation concerned to all others in the network. Thus, maintaining and updating the system graph for global interactions, at each installation, requires a low level of continual communication.

The on-line detection of deadlocks need be considered only for the following complete set of process-resource interactions: (a) a new process enters the system; (b) a new data resource is accessed; (c) a process runs to completion and releases data resources held;* (d) a process in the system requests access to a data resource held by another process; and (e) a data resource held by a process is pre-empted from it.

5.1 Resolution of Process-resource interactions

(a) *A new process enters the system* and/or (b) *a new data resource is accessed*: a new process and/or data resource entry into the system introduces the respective nodes into G_s. An RPA edge is added to G_s whenever a new data resource is accessed either by an entering process or by one in the system. The request by an entering process for a data resource in the system may not be granted, thus introducing a PRW edge. In either case, a deadlock-free system continues to be so.

Assertion 1

If the system in a network configuration is deadlock-free, a new process entry into the system does not lead to deadlock in a circular wait condition. □

*In order to retain the strong consistency result of Eswaran et al., (16) which requires that the processes be "well-formed" and "two-phase," a process is required to be divided into growing and shrinking phases. The first unlock action signals the beginning of the shrinking phase, after which a process cannot issue a lock request on any entity in the database until all entities have been released. The actual implementation of a two-phase protocol (as in SYSTEM R) is to release all data resources held, at the completion of the process. (Private Communication from J.N. Gray, IBM Research Laboratory, San Jose, Calif., USA).

Assertion 2
If the system in a network configuration is deadlock-free, accessing a new data resource does not lead to deadlock in a circular wait condition. □

(c) *A process in the system runs to completion and releases all data resources held*: the process node, and all released data resource nodes which have no requests, are deleted from G_s. A released data resource being waited for by a single process is allocated to the corresponding process. However, an allocation decision for a released data resource with multiple waiting-access requests is done in a manner indicated in Example 5.1. Allocation is done according to the condition shown in Lemma 1 to avoid a potential deadlock.

Assertion 3
If the system in a network configuration is deadlock-free, then neither releasing the data resources held by a completed process for which there are no requests, nor allocating the released data resources for which there is a single waiting-access request leads to deadlock in a circular wait condition. □

Example 5.1
For the configuration of figure 1, let us assume that P_2 requests access to D_1 held by P_1, thus introducing the PRW edge (P_2, D_1) in G_s. Let P_3, which holds D_3, D_4, and D_6, run to completion. D_6 has no requests and hence is deleted from the system graph. D_3 is being waited for by P_1 and is allocated to P_1, whereas D_4 has two waiting-access requests from P_1 and P_4. Assume that P_4 issued its request before P_1 did. If the allocation of D_4 to P_4 is done in a FIFO manner, then processes P_1, P_2, and P_4 will be deadlocked. It is obviously more advantageous to make the allocation of D_4 to P_1 and let P_1 proceed than to make the allocation of D_4 to P_4 and be deadlocked. This is crucial, especially when rollback and recovery in a network environment are expensive. Therefore, in Lemma 1, we give a necessary and sufficient condition to recognize such a situation and to avoid deadlock accordingly. Corollary 3 to Lemma 1 states that in the case of a deadlock-free system with multiple processes waiting for access to a released data resource, there exists at least one process such that an allocation made to this process maintains the system deadlock-free. In the case of multiple processes waiting in a FIFO manner for access to a released data resource, the allocation is made to the first process which maintains the system deadlock-free. Further improvement may be possible by allocating the resource to the first process which not only maintains the system deadlock-free, but also has minimum waiting-access requests on other data resources.

Lemma 1
Let P_i and P_j be any two processes in a deadlock-free system with waiting-access requests to the data resource D_k. The allocation of the data resource D_k to P_i causes deadlock in a circular wait condition if and only if P_j belongs to the reachable set of P_i before the allocation. □

Corollary 3
Let $\{P_i\}_{1 \leqslant i \leqslant n}$ be the processes in a deadlock-free system with waiting-access requests to the data resource D_k. There exists at least one process P_s $(1 \leqslant s \leqslant n)$ such that P_i does not belong to the reachable set of P_s for all $i = 1, 2, \ldots, n$. □

The situation in Example 5.1 arises because process P_4 has waiting-access requests for both D_2 and D_4. In this case our scheme detects a potential deadlock and avoids it accordingly, by virtue of Lemma 1. The potential for a query to be waiting for access to two data resources is illustrated in the Appendix. It is unrealistic to restrict a process to have only one outstanding request, yet this has been the case in approaches by earlier authors – including the one by Goldman.[23] Thus, our approach combines detection and avoidance principles in deadlock handling, and deals with multiple waiting requests in a realistic way.

(d) *A process in the system requests access to a data resource held by another process*: since the request cannot be granted, the introduction of the PRW edge can lead to a cycle in G_s and thus a deadlock. The reachable sets are updated appropriately, and a test for deadlock is carried out.

(e) *Pre-emption of data resources held by a process*: pre-empting data resources is done when a process is aborted in an attempt to break a system deadlock. In such a case, the waiting-access requests of the aborted process are dropped, and the data resources held by the process are released.

Typical criteria for the selection of process(es) to be aborted are outlined below. However, the algorithms for selecting the process are non-trivial, and are not dealt with here.

(i) Abort a process which holds a minimum number of data resources for exclusive access (preferably none), since this can result in reduced rollback costs.
(ii) Of all the deadlocked processes, abort the one which has used minimum CPU time.
(iii) Abort a process which involves rollback at a single installation, in preference to termination of one that leads to global rollback and consequent communication overhead.
(iv) Abort a process which has modified as few data resources as possible, and has interacted with other processes as little as possible, to minimize the cost of rollback.

5.2 *"On-line" deadlock detection algorithms*

Bayer[26] has presented and analysed an on-line transitive closure algorithm for deadlock discovery in databases. To our knowledge on-line deadlock detection algorithms for distributed DBMS have not yet been proposed. In this section, we present procedures for updating reachable sets as interactions go on in the system. Further, these procedures are used in developing an on-line deadlock detection technique.

Procedure ADD, presented below, updates the reachable sets and the system graph when a new edge from N_i to N_j is added to the system.

Procedure ADD (N_i, N_j): (1) for any new node in $\{N_i, N_j\}$, assign its reachable set to null; (2) Assign the reachable set of N_i to the UNION of $\{N_j\}$, and the reachable sets of N_i and N_j; (3) if N_i is not a new node, then for every node N in G_s whose reachable set contains N_i, assign the reachable set of N to the UNION of the reachable sets of N and N_i; (4) update G_s by including new nodes (if any), and the edge from N_i to N_j.

Although it is uncomplicated to update the reachable sets when an arbitrary edge is added to G_s, it seems almost impossible to do so when an arbitrary edge is deleted from G_s. No better method than recalculating the reachable sets seems feasible. On close examination, however, it becomes apparent that the only times when edges are deleted from G_s are: (i) when a process runs to completion and releases data resources; and (ii) when a deadlock is discovered and at least one of the processes involved must be aborted, implying that all data resources held by aborted process(es) will be released, and that all access requests from the process(es) are to be dropped. In case (i) the process is obviously not blocked and hence the corresponding process node in G_s is a sink. Thus, the edges dropped are only those that are directed to a sink. This is a very simple case, and an algorithm can be devised to update the reachable sets. In the deadlock situation of case (ii), however, the processes involved are not sinks in G_s. Therefore, aborting a process and rolling it back requires recalculation of the reachable sets. Maintenance of these sets by incremental updates considerably decreases the chances of synchronization error and that of the problem of system graph becoming obsolete in all the interactions (a) to (d) discussed in Section 5.1. However, the reconstruction of reachable sets for interaction (e) does increase the probability of synchronization error due to system graph obsoleteness.

We present Procedure DELETE to update the reachable sets of all nodes, given that an edge from N_k to N_m is deleted, where N_m is a sink. In updating G_s the edge is deleted, but the appropriate node deletions are done elsewhere (in Procedure ON-LINE_DETECT).

Procedure DELETE (N_k, N_m): (1) for every node N in G_s whose reachable set contains N_m, delete N_m from the reachable set of N; (2) update the set of edges of G_s by deleting the edge from N_k to N_m.

We now propose Procedure ON-LINE_DETECT to handle all deadlock cases discussed in Section 5.1. Procedures ADD and DELETE are extensively used in Procedure ON-LINE_DETECT. Step S1 of the procedure updates reachable sets for a new process and/or data resource entry. Step S2 deals with the case in which a process runs to completion and releases all data resources held. In S2a, the reachable sets are updated by deleting edges corresponding to the release of data resources. Step S2b deletes the nodes in the system graph for released data resources without any requests. Allocation of released data resources with single waiting-access requests is done in S2c. In step S2d, for a set of processes $\{P_k\}$ waiting for a released data resource, the condition in Lemma 1 is tested successively between pairs of processes until a process is found whose reachable set contains no other process in $\{P_k\}$. The data resource should be allocated to such a process to avoid a potential deadlock. In step S3, the case of a process requesting access to a data resource held by another process is dealt with. Step S4 handles the case in which a process is aborted to break a deadlock. In step S5 we carry out a test for the existence of deadlock, for the cases dealt with in steps S3 and S4.

The formal presentation of the technique is provided in reference (17). For the purpose of this paper, on-line deadlock detection is illustrated by utilizing the network configuration of figure 1. We assume the introduction of the PRW edge (P_2, D_1), consequent to P_2 requesting access to D_1, and the RPA edge (D_7, P_5) as a result of P_5 requesting shared access to D_7 (which is also held for shared access by P_6).

Illustration of procedure ON-LINE_DETECT

S1 [A process enters the system, or a new data resource is accessed or both]. Let the new edge added be: (P_0, D_1) (new process entry), or (D_0, P_2) (new data resource entry), or (D_0, P_0) (both). Call Procedure ADD (P_0, D_1), or ADD (D_0, P_2), or ADD (D_0, P_0) as appropriate. STOP.

S2 [A process in the system runs to completion and releases all data resources held]. Let the process which runs to completion be P_3, and the data resources released be D_6, D_3, D_4.

S2a [Update reachable sets by deleting each edge]. Invoke Procedures DELETE (D_6, P_3), DELETE (D_3, P_3), DELETE (D_4, P_3); and remove P_3 from G_s.

S2b [Update G_s by removing released data resource nodes without any waiting-access requests]. Remove D_6 from G_s.

S2c [Allocate released data resources with a single waiting-access request]. Call Procedures DELETE (P_1, D_3), and ADD (D_3, P_1).

S2d [Allocate released data resources with multiple waiting-access requests]. For the released data resource D_4 the condition in

Lemma 1 dictates its allocation to P_1 rather than to P_4. Call Procedures DELETE (P_1, D_4),* ADD (P_4, D_4), and ADD (D_4, P_1). STOP.

S3 [A process in the system requests access to a data resource held by another process]. Let the process be P_8, and the data resource be D_8. Call Procedure ADD (P_8, D_8). GO TO STEP S5.

S4 [A data resource held by a process is pre-empted from it, to break a deadlock]. Let the aborted process be P_5. Remove from G_s the edges (D_8, P_5), (D_7, P_5), and (P_5, D_9). Calculate the reachable sets of all nodes of G_s. GO TO STEP S5.

S5 [Detect deadlock]. Apply second step of the method outlined in Section 4 to detect deadlocks (if any). If deadlocked, then determine which process is to be aborted and apply Step S4, else STOP.

6 Highlights of Our Proposal

It is difficult to estimate the performance effects of deadlock detection or deadlock prevention in distributed DBMS, since communication time is critical. Because of the complexity of distributed DBMS a significant factor in handling deadlocks would be operational efficiency. But the communication aspects make it impractical to estimate the performance of such algorithms analytically. Once distributed DBMS becomes a commercial reality, experimental data can be gathered to measure the performance. Nevertheless, our proposal has several advantages elaborated below:

- In the deadlock detection approach proposed in this paper, the communication needs are quite modest. Every time a data resource is allocated to a process, or the process is made to wait, or when a process releases a data resource, the access controller at the concerned installation sends that information to all other installations in the network. Thus, maintaining and updating the system graph at each installation as global interactions go on in the system is relatively easy, compared to the necessity of transmitting large tables among installations – as was the case in the approaches by Chu and Ohlmacher,[18] Maryanski,(19) Chandra et al.,(20) Mahmoud and Riordon,(21,22) and Goldman.(23)
- The technique of deadlock detection suggested in this paper (Section 4) identifies the processes directly responsible for the deadlock (Corol-

*This has the effect of deleting D_4 from the reachable set of P_4, and those of the nodes from which D_4 was accessible. Thus, the procedure ADD (P_4, D_4) is *mainly* called to correct the reachable sets, even though the edge (P_4, D_4) is already present in the system graph.

lary 1). It is also possible for our method to identify a process blocked forever but not deadlocked, by virtue of its waiting for a process which is involved in a deadlock (Corollary 2).

- Processes are never delayed by our technique, because a process whose request for a data resource can be granted may proceed without waiting for the deadlock detection mechanism. The database administrator can design a scheme to invoke the detection mechanism for every X units of time, or for every Y accesses granted, or for every Z accesses not granted, or any combination of these.
- In our approach, a process may have any number of outstanding requests simultaneously. For most others, including that of King and Collmeyer(12) and Goldman,(23) a process is restricted to having at most one outstanding request. In real-world applications this restriction is not practical.
- In the case of a number of readers sharing a data resource, our algorithm does not require any special scheme like that of Goldman in which one different copy of the OBPL is formed for each such reader. In the case of shared access his approach leads to a heavy overhead in computation and in communication.
- Our proposal deals with every request in the same manner, and can be considered a unified approach, since the detection technique does not classify requests according to the relationship between the origin of the process and the installation of residence of the data resource accessed. Whereas, in all other approaches,(20–23) the algorithms deal with each access request according to the classification of the request.

7 Conclusions

Today, distributed DBMS is the centre of intense research and development activity in both the academic and the industrial worlds, as reported recently by Ziegler,(27) a special issue of IEEE *Computer*(28), and Rothnie et al.(29)

To accelerate the implementation of distributed DBMS, it is necessary to instill in users confidence that their application jobs will not be held up at remote locations indefinitely due to deadlock. Without such confidence, there may be a lack of acceptance of distributed DBMS in a data processing environment, which in turn will impede further progress. The on-line deadlock detection technique proposed in this paper is a step towards increasing that user confidence.

If a deadlock is detected, it must be broken by aborting and rolling back at least one process. An effective and efficient rollback and recovery mechanism is of immense need in distributed DBMS. Such a mechanism can make our approach to deadlock detection much more attractive. Research efforts are planned in this direction.

Acknowledgments

We thank Anne Brindle for her numerous suggestions in improving the comprehensibility of an earlier draft of this paper. Her generous help is recorded with deep appreciation.

We also thank the referees and the editor for identifying possible misinterpretations and offering helpful clarifications.

Appendix

To show the possibility of more than one outstanding request per process, we choose Chamberlin's "Presidential data base"(30) in Codd's relational model of data.(31) The relations in the database are in Codd's Third Normal Form.(32) A query to the database, which causes two simultaneous requests, is expressed in Codd's *Data Sub-Language ALPHA*.(33)

The relations in the database and the query are:

```
R1  ELECTIONS_WON (YEAR, WINNER_NAME,
                         WINNER_VOTES)
R2  PRESIDENTS (NAME, PARTY, HOME_STATE)
R3  ELECTIONS_LOST (YEAR, LOSER_NAME,
                          LOSER_VOTES)
R4  LOSERS (NAME, PARTY)
```

Query

(a) (In English) List the election years in which a Republican from Illinois was elected.

The intent of this query is: Retrieve the YEAR attribute from the tuple of relation ELECTIONS_WON, whose WINNER_NAME attribute matches the NAME attribute of a tuple of relation PRESIDENTS, provided the tuple of relation PRESIDENTS has a HOME_STATE of ILLINOIS and a PARTY equalling REPUBLICAN.

(b) (In DSL ALPHA)

```
RANGE PRESIDENTS P
RANGE ELECTIONS_WON E
GET W E. YEAR: ∃ P (P. NAME = E. WINNER_NAME &
                    P. PARTY = 'REPUBLICAN' &
                    P. HOME_STATE = 'ILLINOIS')
```

For the execution of this query it is essential to gain access to the relations R2 and R1 at the same time, causing two potential outstanding requests.

References

(1) G.M. Booth, "Distributed information systems," *Proc. AFIPS National Computer Conference*, 45, June 1976, pp. 789–794.

(2) P.H. Enslow, Jr., "What is a 'distributed' data processing system?" IEEE Computer, vol. 11, January 1978, 13–21.

(3) S.K. Chang, "A model for distributed computer system design," IEEE *Transactions on Systems, Man, and Cybernetics, SMC-5*(6), May 1976, pp. 344–359.

(4) R.H. Eckhouse, Jr., J.A. Stankovic, and A. van Dam, "Issues in distributed processing – an overview of two workshops," IEEE Computer, vol. 11, January 1978, 22–26.

(5) B.P. Lientz and I.R. Weiss, "Trade-offs of secure processing in centralized *versus* distributed networks," Computer Networks, vol. 2, February 1978, 35–43.

(6) T.A. Marsland and S.F. Sutphen, "Design and experience with a distributed computing system," *Proc. Canadian Computer Conference, CIPS Session* 78, Edmonton, Canada, May 1978, pp. 367–371.

(7) J.P. Fry and E.H. Sibley, "Evolution of data-base management systems," Computing Surveys, vol. 8, March 1976, 7–42.

(8) R. Peebles and E. Manning, "System architecture for distributed data management," IEEE Computer, vol. 11, January 1978, 40–47.

(9) F.J. Maryanski, "A survey of developments in distributed data base management systems," IEEE Computer, vol. 11, February 1978, 28–38.

(10) R.C. Holt, "Some deadlock properties of computer systems," Computing Surveys, vol. 4, September 1972, 179–196.

(11) A. Shoshani and A.J. Bernstein, "Synchronization in a parallel accessed data base," CACM, vol. 12, November 1969, 604–607.

(12) P.F. King and A.J. Collmeyer, "Database sharing: an efficient mechanism for supporting concurrent processes," *Proc. AFIPS National Computer Conference*, 42, June 1973, pp. 271–275.

(13) D.D. Chamberlin, R.F. Boyce, and I.L. Traiger, "A deadlock-free scheme for resource locking in a data-base environment," *Information processing* 74, *Proc. IFIP Congress*. Amsterdam: North-Holland Publishing Co., August 1974, pp. 340–343.

(14) G. Schlageter, "Access synchronization and deadlock analysis in database systems: an implementation-oriented approach," Information Systems, vol. 1, 1975, 97–102.

(15) R.E. Stearns, P.M. Lewis, II, and D.J. Rosenkrantz, "Concurrency control for database systems," *Proc.* IEEE *17th Annual Symposium on Foundations of Computer Science*, October 1976, pp. 19–32.

(16) K.P. Eswaran, J.N. Gray, R.A. Lorie, and I.L. Traiger, "The notions of consistency and predicate locks in a data base system," CACM, vol. 19, November 1976, 624–633.

(17) S.S. Isloor and T.A. Marsland, "Deadlock detection in databases distributed on a network of computers," Tech. Rep. TR 78-3, Dept. of Computing Science, University of Alberta, Edmonton, Alberta, Canada, May 1978. (40 pages)

(18) W.W. Chu and G. Ohlmacher, "Avoiding deadlock in distributed data bases," *Proc. ACM National Conference*, 1, November 1974, pp. 156–160.

(19) F.J. Maryanski, "A deadlock prevention algorithm for distributed data base management systems," Technical Report CS 77-02, Computer Science Dept., Kansas State University, Manhattan, Kansas, February 1977. (24 pages)

(20) A.N. Chandra, W.G. Howe, and D.P. Karp, "Communication protocol for deadlock detection in computer networks," IBM Technical Disclosure Bulletin, vol. 16, March 1974, 3471–3481.

(21) S.A. Mahmoud and J.S. Riordon, "Protocol considerations for software controlled access methods in distributed data bases," *Proc. International Symposium on Computer Performance Modeling, Measurement and Evaluation*, Cambridge, Mass., March 29–31, 1976, pp. 241–264.

(22) S.A. Mahmoud and J.S. Riordon, "Software controlled access to distributed data bases," INFOR, vol. 15, February 1977, 22–36.

(23) B. Goldman, "Deadlock detection in computer networks," Technical Report MIT/LCS/TR-185, Laboratory for Computer Science, M.I.T., Cambridge, Mass., September 1977. (180 pages)

(24) E.G. Coffman, Jr., M.J. Elphick, and A. Shoshani, "System deadlocks," Computing Surveys, vol. 3, June 1971, 67–78.

(25) G. Le Lann, "Pseudo-dynamic resource allocation in distributed databases," *Proc. Fourth International Conf. on Computer Communications, ICCC-78,* Kyoto, Japan, September 1978, pp. 245–251.

(26) R. Bayer, "Integrity, concurrency, and recovery in databases," in *Lecture notes in computer science 44, Proc. ECI Conference 1976,* K. Samelson (ed.). Berlin: Springer-Verlag, 1976, pp. 79–106.

(27) K. Ziegler, Jr., "Distributed data base, where are you? – (a tutorial)," *Information processing 77, Proc. IFIP Congress.* Amsterdam: North-Holland Publishing Co., August 1977, pp. 113–115.

(28) A. van Dam and J. Stankovic (Guest Editors), "Distributed processing" (special issue), IEEE Computer, vol. 11, January 1978, 10–57.

(29) J.B. Rothnie, N. Goodman, and T. Marill, "Database management in distributed networks" (Chapter 10), *Protocols and techniques for data communication networks,* Franklin F. Kuo (ed.). Englewood Cliffs, NJ: 1979 (in press).

(30) D.D. Chamberlin, "Relational data-base management systems," Computing Surveys, vol. 8, March 1976, 43–66.

(31) E.F. Codd, "A relational model of data for large shared data banks," CACM, vol. 13, June 1970, 377–387.

(32) E.F. Codd, "Normalized data base structure: a brief tutorial," *Proc. 1971 ACM-SIGFIDET Workshop on Data Definition, Access, and Control,* November 1971, pp. 1–17.

(33) E.F. Codd, "Relational completeness of data-base sublanguages," *Courant computer science symposia 6, Data Base Systems.* Englewood Cliffs, NJ: Prentice-Hall, 1971, pp. 65–98.