A SURVEY OF DISTRIBUTED DEADLOCK DETECTION ALGORITHMS

Ahmed K. Elmagarmid

Computer Engineering Program
Department of Electrical Engineering
Pennsylvania State University
University Park, PA 16802
(814) 863-1047

Abstract

This paper surveys research work performed within the last five years in distributed deadlock detection. The last survey paper on this topic appeared in 1980; since that time a large number of interesting algorithms have been discribed in the literature. A new, more efficient scheme is the probe-based deadlock detection strategy used by many of the new algorithms. This paper will concentrate on distributed deadlock detection algorithms. Only detection of resource deadlocks will be reviewed here, though other types of deadlock handling strategies and environments are briefly mentioned.

1.0 INTRODUCTION

In modern computer systems, several transactions may compete for a finite number of resources. Upon requesting a resource, a transaction enters a wait state if the request is not granted due to non-availability of the resource. A situation may arise wherein waiting processes may not ever get a chance to change their states. This can occur if the requested resources are held by other waiting processes. This situation is called deadlock.

The deadlock problem has been extensively studied in database and operating systems. Numerous algorithms on centralized deadlock detection [ELMAGARMID85], [HO], [MAHMOUD76], [MAHMOUD77], distributed deadlock detection [BADAL], [BITTMAN], [BRACHA84], [BRACHA85], [CHANDY82], [ELMAGARMID85], [GOLDMAN], [HAAS81], [HAAS83], [HERMAN], [HO], [ISLOOR79a], [ISLOOR79b], [ISLOOR80], [JAGANNATHAN82], [JAGANNATHAN83], [KAWAZU], [MAHMOUD76], [MAHMOUD77], [MARSLAND], [MENASCE78], [MENASCE79], [MITCHELL], [MOSS], [OBERMACK80a], [OBERMACK80b], [OBERMACK82], [SINHA84], [SINHA85], [TSAI82a], [TSAI82b]; prevention or avoidance algorithms [ANDREWS], [JORDAN], [KORTH82], [LOMET78a], [LOMET78b], [LOMET80], [MINOURA], [ROSENKRANTZ]. Communication deadlocks have not, however, been studied as extensively. Few algorithms have been published to detect deadlocks among communicating entities [RAEUCHLE], [NATARAJAN] [CHANDY83]. Two annotated bibliographies can be found in the literature [NEWTON], [ZOBEL], and the last thorough survey paper is by Isloor and Marsland [ISLOOR80].

Deadlocks in a system can be handled in three ways:

1) Prevention: Guaranteeing that deadlocks can never occur in the first place. This requires no run time support.
2) Avoidance: Detecting potential deadlocks in advance and taking action to insure that deadlock will not occur. This requires run time support.
3) Detection: Allowing deadlocks to form and then finding and breaking them. As in the avoidance scheme, this requires run time support.

1.1 Deadlock Prevention:

In deadlock prevention all the resources which may be needed by a transaction must be predeclared. In this strategy, a request is granted only if all the resources it needs are available and the system can guarantee that none of these resources will be needed by any ongoing transaction. In other words, all resources needed are reserved in advance. However, they need not be allocated a priori.

Deadlock prevention has two obvious disadvantages: First, preallocation of resources leads to reduced concurrency. Second, evaluation of the safety of the request results in additional overhead. A major advantage of this scheme is that it does not involve any transaction rollback or restart due to deadlocks. Prevention is the only feasible scheme for handling deadlock in systems that have no provision for restoring states. But since the ability to undo work is generally necessary in

systems that are tolerant to failures, the above-mentioned advantage may be irrelevant in many situations. Deadlock prevention is considered impractical except for systems which have a predefined structure. A large number of prevention algorithms have been given in the literature [COFFMAN], [ANDREWS] or see the annotated bibliography by Zobel [ZOBEL].

1.2 Deadlock Avoidance:

In deadlock avoidance, transactions need not determine the resources they require a priori. Transactions are allowed to proceed unless a requested resource is unavailable. In case of a conflict, transactions may be allowed to wait for a fixed time interval with the expectation that the resource may become available during that interval. In case the resource does not become available, either the requesting transaction or the holding transaction may be aborted. The victim selection criteria for the abortion of a transaction vary depending on the avoidance scheme used. Two well referenced avoidance schemes that make use of priorities of transactions are Wound-Wait and Wait-Die [ROSENKRANTZ].

Though deadlock avoidance may abort transactions unnecessarily, it is considered more attractive than prevention in database systems since such systems already have the ability to abort transactions.

1.3 Deadlock Detection

In systems where deadlock detection strategies are used, conflicts resulting from requests for resources are handled by allowing the requesting transactions to wait freely. As a result, deadlocks may arise and therefore must be detected and resolved. The main task performed by a detection algorithm is to find cycles among transactions each waiting for a resource held by the other. In essence, deadlock detection consists of finding cycles in a directed graph. In the graph, transactions and resources are represented by vertices, and the requests and allocations by edges.

Deadlock detection algorithms can be distributed, centralized or hierarchical. In distributed schemes, no single site has all the information relating to all transactions and resources involved in the graph. Therefore, to detect deadlocks, information must be passed between sites. A variant of this scheme, an edge chasing deadlock detection algorithm, involves sending information according to the structure of the graph [MOSS].

Centralized deadlock detection algorithms require that all information represented by the graph be kept at the acting controller, which is responsible for running the deadlock detection and resolution algorithms. Hierarchical deadlock schemes are based on providing several levels of hierarchy, namely, local, regional and global. The hierarchy should be established so that deadlocks can be detected by a site as close to the sites involved in the deadlock as possible [MENASCE].

The main disadvantage of detection schemes is the additional overhead incurred due to detection of cycles in the graph and abortion and restart of transactions upon detection of deadlocks. The distributed detection strategies may have dditional overhead due to the inter-site message transfers. Selection of the transaction to be aborted adds to the complexity of the scheme. In some algorithms based on this strategy, a situation may arise wherein more than one site detects the same deadlock [TSAI86]. This complicates the deadlock resolution phase, which must be performed once deadlock is detected.

The correctness of a deadlock detection algorithm depends on two conditions. First, all deadlocks must be detected in a finite time. Second, if a deadlock is detected, it must indeed exist. More research must be performed in providing proof techniques suitable for proving the correctness of deadlock algortihms. Proofs provided in the literature to show the correctness of the algorithms are sketchy at best. The lack of suitable proof systems has led to a situation where an algorithm that has been "shown" to be correct by its designers has later been proved to be incorrect; see [GLIGOR] for example. An attempt to use a proof system that is based on axiomatic semantics [SOUNDARARAJAN] has been used to prove the correctness of a deadlock detection algorithm [ELMAGARMID86].

2.0 Survey of Distributed Deadlock Detection Algorithms:

Numerous algorithms in distributed deadlock detection have been proposed in the literature, and they fall into two basic categories. Algorithms that belong to the first category pass information about transaction requests to maintain global wait-for-graph. In the algorithms in the second category, simpler messages are sent among transactions; no global wait-for-graph is explicitly constructed. However, a cycle in the graph will ultimately cause messages to return to the initiator of the deadlock detection message, signaling the existence of deadlock.

As mentioned in the previous section, the correctness of a deadlock detection algorithm depends on two conditions: namely, that all deadlocks must be detected in a finite time, and that if a deadlock is detected, it must indeed exist. Detecting a non-existent deadlock is referred to as a false (phantom) deadlock detection. Wuu and Bernstein [WUU] have shown that no phantom deadlocks will be detected if transactions follow a Two Phase Locking protocol (2PL). Transactions obeying 2PL acquire all locks first and release them one by one [ESWARAN].

In this section, a few algorithms for distributed deadlock detection are surveyed. Their ability to detect all deadlocks without detecting phantom ones is discussed.

2.1 Menasce's Scheme [MENASCE79]:

This algorithm was the first to use a condensed transaction-wait-for graph (TWF) in which the vertices represent transactions and edges indicate dependencies between transactions. In this graph, an edge (T^i,T^j) exists if T^i is blocked on T^j and must wait for T^j to release the resource. This resource need not be the one T^i is waiting for. An edge in TWF represents condensation of dependencies between transactions. A vertex denoting a transaction does not have an outgoing edge if the transaction is not blocked. A blocking set(T) is defined as a set of all non-blocked transactions that can be reached by following a directed path from the node representing transaction T. For each transaction T^1 in the blocking set(T), the pair (T,T^1) is called the blocking pair, the site of origin of a transaction T is denoted by Sorig(T), and the graph at site K by TWF(K). The algorithm is described by the following rules:

Rule 1:

Event: Transaction T requests for a resource r at site S_k, and r is currently held by transactions T_1, T_2, ...T_n.

Action: An edge is added from the node denoting T, to each of the transactions $T_1 \ldots T_n$. If this action causes a cycle in TWF(K), then a deadlock exists. For each transaction T' in blocking set(T), a blocking pair (T,T') is sent to Sorig(T) if Sorig(T) = S^k, and to Sorig(T') if Sorig(T') = Sk.

Rule 2:

Event: A blocking pair (T,T') is received at site S_k.

Action: An edge is added from T to T' in TWF(K). If a cycle results, then a deadlock is detected. If T' is blocked and Sorig(T) = S_k, then for each transaction T'' in the blocking set(T), send blocking pair (T,T'') to Sorig(T'') if Sorig(T'') =S_k.

The algorithm can fail to detect some deadlocks and may discover false deadlocks. Gligor and Shattuck [GLIGOR] proposed modification to the algorithm to fix the first problem. However, the modified algorithm is impractical. This scheme suffers from phantom deadlocks since blocking pairs, which are basically graph update messages, may arrive out of sequence. Consequently, a message requesting deletion of an edge due to release of a resource may be received before the blocking pair containing the request for the resource is received. If the deletion message is ignored, a transaction would become ostensibly blocked. This, in turn, could result in phantom deadlock detection. Because the scheme may incorrectly determine whether a transaction is blocked or not, some deadlocks may go undetected since the status of a transaction can not be determined unless the outcome of a request is known. However, when the outcome is indeed available, it is not used by the algorithm to detect deadlocks.

2.2 Chandy's Scheme [CHANDY82]:

The scheme described in [CHANDY] is specified using two sets of axioms which are also used to derive the correctness proofs. The alogrithm makes no assumptions other than that the messages are received correctly and in order.

This algorithm uses TWF graphs to represent the status of transactions at the local sites and uses probes to detect global deadlocks. Basically, the scheme uses colored graphs whose edges are gray, black or white. An edge (T_i,T_j) is

Gray: If transaction T_i has sent request to transaction T_j, and T_j is yet to receive it.

Black: If transaction T_j has received the request but has not yet sent reply to T_i.

White: If transaction T_j has sent a reply to process T_i, but T_i has not yet received it.

The algorithm by which a transaction T_i determines if it is deadlocked is called a probe computation. A probe is issued if a transaction begins

to wait on another transaction and gets propagated from one site to another based on the status of the transaction that received the probe. If the transaction that received the probe is active, the color of the incoming edge from other transactions will not be black. The coloring scheme allows update of the wait-for graph without inconsistencies. Only the transaction that receives the probe can determine if the color of the incoming edge from the transaction that sent the probe to itself is black. The existence of such an edge indicates that there is indeed a wait-for relation between the transactions. The probe is further propagated only if the edge is determined to be black and there is an outgoing edge that is not white. Although whether the edge is black or gray can not be determined, this clear distinction is indeed not required for probe propagation as the transaction that receives the probe next can distinguish it. In other words, a probe travels only along black edges of the wait-for graph. The probes are meant only for deadlock detection and are distinct from requests and replies. A transaction sends at most one probe in any probe computation. If the initiator of a probe computation gets back the probe, then it is involved in a deadlock. The scheme does not suffer from false deadlock detection even if the transactions do not obey the two-phase locking protocol. This could be achieved through the coloring scheme which helps in keeping the wait-for graphs up to date.

2.3 Obermack's Algorithm [OBERMACK82]:

This algorithm recreates the transaction-wait-for graph (TWFG) each time deadlock detection is performed as in the algorithm in [GOLDMAN].

The deadlock detection algorithm at each site builds and analyzes directed TWFG and uses a distinguished node at each site. This node is called "external" and is used to represent the portion of TWFG that is external (unknown) to the site. The status of each agent's direct communication links at the local site towards its cohorts at other sites determines whether the transaction is waiting for "external" or the "external" is waiting for the transaction's agent at the site. The detection algorithm at each site performs the following steps:

1) Build TWFG.

2) Obtain and add the TWFG information received as strings from other sites to the TWFG.

3) Create wait-for edges from "external" to each node representing agent of transaction that is expected to send on communication link.

4) Create Wait-for edges to "external" from the node representing of transaction that is waiting to receive from communication link.

5) Analyze the TWFG listing all elementary cycles.

6) Select a victim to break each cycle that does not contain the node "external." As each victim is chosen, remove all cycles that include the victim.

7) For each cycle EX--> T_1--> T_2...T_x-->EX' containing the node "external," send a string EX, T_1,T_2...T_x to the site T_x is waiting for to receive, if transaction id of T_1 is greater than that of T_x.

Though this algorithm is frequently referenced, it does not work correctly; it detects false deadlocks because the wait-for graphs constructed do not represent a snap-shot of the global TWFG at any instant. The comparison of transaction id's at step 7 reduces the number of messages sent by one half. The algorithms in [TSAI82a, SINHA84] adopt this strategy for reducing the number of messages required for deadlock detection.

2.4 Bracha's Algorithm [BRACHA84]:

In this paper two algorithms are given, one for dynamically changing systems and the other for a static system where transmission delays are ignored for simplification. The algorithms use a model similar to the AND/OR in which transactions can request any N available from a pool of M resources. An OR request corresponds to N = 1 while an AND request corresponds to N = M.

The static algorithm consists of two phases: A notification phase, in which transactions are notified that a deadlock detection algorithm has started, and a granting phase, in which active transactions simulate granting of requests. Deadlocked nodes are the nodes that are not made active by the second phase. For static systems with messages in transit, the authors have used colored wait-for graph to take into account messages in transit in the communication channels, and to represent a static snapshot of the ongoing activities in the system. The dynamic algorithm is not summarized in this paper.

2.5 Sinha's Scheme [SINHA84]:

This algorithm , an extension to Chandy's [CHANDY82] scheme, is based on priorities of transactions. Using

priorities, the number of messages required for deadlock detection is reduced considerably. An advantage of the scheme is that the number of messages in the best and worst cases can be determined easily.

The authors' model consists of transactions and data managers that are responsible for granting and releasing locks. A transaction's request for a lock on a data item is sent to the data manager of the item. If a request can not be granted, the data manager initiates deadlock computation by sending a probe to the transaction that holds a lock on the data item, if the priority of the holder is greater than that of the requestor. The transaction inserts this probe in a probe-q that it maintains. he probe is then sent to the data manager of the data item it is waiting for. At this stage of deadlock computation, priorities of transactions are used to decide whether to propagate the probe or not. The probe is propagated only if the priority of the holder of the data item it manages is greater than that of the initiator. When a transaction begins to wait for a lock, all the probes from its queue are propagated. When a data manager gets back the probe it initiated, deadlock is detected. Since the probe contains the priority of the youngest transaction in the cycle, the youngest transaction is aborted.

2.6 Mitchell's Algorithm [MITCHELL]

This is an edge chasing algorithm in which each transaction uses a public and private label for deadlock detection. Initially these labels for all transactions have the same value. The scheme is explained in the following four steps:

1) Block Step: When a transaction becomes blocked waiting for a second transaction, both of its labels are incremented to a value greater than that of the blocking transaction.

2) Active Step: A transaction becomes active when it gets a resource, times out or fails, or when the owner of a resource changes.

3) Transmit Step: When a blocked transaction discovers that its public label is smaller than that of the blocking transaction, it changes its label to a value equal to that of the blocking transaction.

4) Detect Step: When a transaction receives its own public label back, a deadlock is detected.

When a transaction begins to wait for a resource held by another transaction, it executes the block step. When a transaction becomes active, it executes the active step. Periodically, the blocked transactions read the public label of the blocking transaction. Based on the ordering between this label and its own public label, the transmit step explained above is executed.
Due to the transmit step, the largest public label migrates in the opposite direction along the edges of the wait-for graph.

This algorithm is easily implemented and does not detect false deadlocks in the absence of process failures. It also can detect all existing deadlocks.

2.7 Ho's Scheme [HO]:

The authors have given three algorithms that use transaction and resource tables. While the first two are discussed here, the third one, the hierarchical algorithm is not discussed in this survey. The transaction table at each site maintains information regarding resources held and waited on by local transactions. The resource table at each of the sites maintains information regarding the transactions holding and waiting for local resources. Periodically, a site is chosen as a central controller responsible for performing deadlock detection.

In the first algorithm, two-phases are used to determine whether a deadlock exists or not. Because deadlocks persist once they have occured, the second phase is basically used to verify the findings of the first phase. In the first phase, the central controller broadcasts a message to all sites, requesting them to send in their transaction tables. Upon receipt of these tables, the controller constructs a wait-for graph based on information obtained from the tables. If a cycle is detected, a message is again broadcast to all sites in the system requesting them to send in their tables again. The decision regarding the presence of any deadlocks is validated in the second phase by constructing a wait-for graph using only the transactions reported in both phases. This algorithm does not employ resource tables for deadlock detection.

The drawback of this scheme is that it requires 4n messages, where n is the number of sites in the system. An obvious improvement to the algorithm can be achieved by sending messages in the second phase only to those sites involved in the deadlock cycle, thereby reducing the number of messages required to detect deadlocks. A counter example was given

by Jagannathan and Vasudevan [JAGANNATHAN82b] to show that the algorithm detects false deadlocks. The transactions in the counter example do not obey the two-phase locking protocol.

In the one-phase algorithm, each site maintains resource and transaction tables. The central controller periodically calls for transaction and resource status tables. The wait-for graph is constructed using the transactions for which identical entries exist in both of these tables, ensuring that the information regarding these transactions is indeed the latest.

2.8 Kawazu's Algorithm [KAWAZU]:

The authors divide this algorithm into two phases. In the first phase local deadlocks are detected, and in the second phase global deadlocks are detected in the absence of local deadlocks. The local deadlock detection step is initiated when a transaction begins to wait for a resource. If no local deadlock is detected, the detection of possible global deadlocks begins. To detect global deadlock, the local wait-for graphs are gathered to construct a pseudo-wait-for graph. This graph does not necessarily represent the true status of transactions.

This scheme suffers from phantom deadlocks, because each local wait-for graph is not collected at the same time due to communication delays. Also, in case a transaction simultaneously waits for more than one resource, some global deadlocks may go undetected since the global deadlock detection is initiated only if no local deadlock is detected. For a counter example to this algorithm see [JAGANNATHAN82b].

2.9 Haas's Scheme [HAAS83]:

This scheme combines the approaches of [CHANDY82] and [OBERMACK82]. In this scheme, when a transaction T that is waited on by other transactions begins to wait for a remote resource, it triggers a deadlock computation by sending messages to the transactions it is waiting for. These messages convey information about potential deadlock cycles as in [OBERMACK82]. The message is initially a set R that contains T and the set of transactions waiting on T.

When a site receives such a message, the controller at the site checks if the transaction for which the message has been received is active. The message is ignored if the transaction is found to be active. In the event of the transaction waiting for resources, the transaction name's presence is looked for in each member of R. If the above search is successful, a deadlock has been detected and the deadlocked sets are deleted from the set R. The transactions name is then appended to all remaining elements of R. The appended set is then propagated to the sites of resources the transaction has been waiting for, because a transaction may be involved in more than one deadlock at any time. The algorithm results in only one process detecting a deadlock cycle.

3.0 CONCLUSION:

The scheme of Menasce and Muntz [MENASCE79] requires at each site storage proportional to the size of local transaction wait-for graph. The edges of this graph are both direct and indirect (condensed). Though the condensed edges require extra memory, it may not be substantial in most cases. The number of messages required for deadlock detection may become to be exponential. As mentioned before, the scheme does not detect all deadlocks and detects false deadlocks. The scheme of [CHANDY82] does not suffer from these disadvantages and is well suited for proving correctness of the algorithm; the disadvantage in the scheme is that a process that detects a deadlock , as in [MENASCE79], is not aware of all the transactions involved in the cycle, information required for efficient deadlock resolution. The scheme of [SINHA84] alleviates this problem by providing a field for the victim to be chosen to break the deadlock. The scheme of [HAAS83] gives the deadlock detector information regarding transactions involved in the cycle and requires no message for detecting deadlocks involving only two transactions. Though the messages in the scheme of [HAAS83] are longer than that of [CHANDY82], they are not longer than the notification messages of [CHANDY82]. While the worst case complexity of [CHANDY82] is fourth power of n, that of [SINHA84] is second power of n but the latter algorithm requires that transactions obey the two-phase locking protocol. For deadlock cycles involving fewer than seven transactions, the scheme of [HASS83] outperforms the scheme of Chandy and Misra. As per [GRAY81], most deadlocks involve two to four transactions. Hence the scheme of [HAAS83] is preferable. The schemes of [BRACHA84] and [MITCHELL] also use short message probes as in [CHANDY82] and have the advantages and disadvantages of that scheme. The schemes of [OBERMACK82] ,

[HAAS83] and [KAWAZU] have long deadlock messages because of the wait-for graph information sent as part of these messages. However, these messages are helpful during deadlock resolution phase. The scheme of [OBERMACK82], though implemented in system R*, suffers from false deadlock detection. The scheme of [KAWAZU] involves sending local sub-wait-for graphs for deadlock detection as in [OBERMACK82]; the deadlock resolution will become complicated since more than one site can detect the same deadlock. While the one-phase scheme of [HO] does not detect false deadlocks, it requires all sites to send in their wait-for graph information. This results in superfluous information sent to the central controller, resulting in extra inter-site message overhead; also, since the deadlock detection is performed periodically by the central controller, the time required for deadlock detection depends on the frequecy of such calls to the sites.

While most of the work in this area has concentrated on the design of new algorithms. More research in the specification, verification and performance evaluation of deadlock detection algortihms is needed. The lack of unified means by which researchers may specify their algorithms resulted in most of these algorithms being shown incorrect. Almost no work exists in the area of evaluating the performance of these algortihms.

REFERENCES

[ANDREWS] Andrews G.R. and Levin G.M., "On-The-Fly Deadlock Prevention". ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing 1982. 165-172.

[BADAL] Badal D.Z. and Gehl M.T., "On Deadlock Detection in Distributed Computing Systems". IEEE INFOCOM 1983.

[BITTMAN] Bittman P. and Unterauer K., "Models and Algorithms for Deadlock Detection", in Opearting Systems: Theory and Practice. D. Lanciaux editor. North Holland Publishing Company 1979. 101-111.

[BRACHA84] Bracha G. and Toueg S., "A Distributed Algorithm for Generalized Deadlock Detection", ACM Symposium on Principles of Distributed Computing. 1984. (Also appeared as Tech Report TR 83-558 June 1983, Department of Computer Science Cornell University).

[BRACHA85] Bracha G., "Randomized Agreement Protocols and Distributed Deadlock Detection Algorithms", Ph.D. Thesis, Department of Computer Science, Cornell University. Jan 1985. (Also, TR 85-657)

[CHANDY82] Chandy K.M. and Misra J., "A Distributed Algorithm for Detecting Resource Deadlocks in Distributed Systems", ACM Symposium on Principles of Distributed Computing. 1982. 157-164.

[CHANDY83] Chandy K.M., Misra J. and Haas L.M., "Distributed Deadlock Detection", ACM Transactions on Computer Systems Vol 1 No. 2 May 1983. 144-156.

[COFFMAN] Coffman E.G., Elphic M.J. and Shoshani A., "System Deadlocks," Computing Surveys, Vol 3, No. 2, June 1971, 67-78.

[ELMAGARMID85] Elmagarmid A.K.,"Deadlock Detection and Resolution in Distributed Processing Systems," Ph.D. Thesis. The Ohio State University 1985.

[ELMAGARMID86] Elmagarmid A.K., Soundararajan N. and Liu M.T.," A Distributed Deadlock Detection Algorithm and Its Correctness Proof," Submitted for publication. 1986.

[ESWARAN] Eswaran K.P., et al, "The Notions of Consistency and Predicate Locks in a Database System," Communications of the ACM, Vol. 19, No. 11, Novemnber 1976, 624-633

[GLIGOR] Gligor V.D. and Shattuck S.H., "On Deadlock Detection in Distributed Systems", IEEE Transactions on Software Engineering Vol SE-6, Sep 1980. 435-440. (Also, Tech Report 837, Department of Computer Science, University of Maryland Dec 1979).

[GOLDMAN] Goldman B., "Deadlock Detection in Computer Networks", MIT/LCS/TR-185. Sep 1985.

[GRAY81] Gray J., Homan P., Korth H. and Obermack R., "A Straw Man Analysis of the Probability of Waiting and Deadlock in a DB System" RJ3066 IBM Research Laboratory. San Jose, California. February 1981.

[HAAS81] Haas L.M., "Two Approaches to Deadlock in Distributed Systems", Ph.D. Thesis. Department of Computer Science. The University of Texas at Austin. August 1981.

[HAAS83] Haas L.M. and Mohan C., "A Distributed Deadlock Detection Algorithm for a Resource-Based System" RJ3765 1/25/83 IBM Research Laboratory, San Jose, CA 95193.

[HERMAN] Herman T. and Chandy K.M.," A Distributed Procedure to Detect AND/OR Deadlock", TR LCS-8301 Department of Computer Science, The University of Texas at Austin February 1983.

[HO] Ho G.S. and Ramamoorthy C.V., "Protocols for Deadlock Detection in Distributed Database Systems", IEEE Transactions on Software Engineering, Vol SE-8 No. 6, November 1982. 554-557.

[ISLOOR79a] Isloor S.S., "Consistency Aspects of Distributed Databases". Ph.D. Thesis Department of Computer Science, The University of Alberta August 1979. TR79-4

[ISLOOR79b] Isloor S.S. and Marsland T.A., "System Recovery in Distributed Databases", IEEE COMPSAC 1979 421-426.

[ISLOOR80] Isloor S.S. and Marsland T.A., "The Deadlock Problem: An Overview", Computer, Sept 1980, 58-70.

[JAGANNATHAN82a] Jagannathan J.R. and Vasudevan R., "A Distributed Deadlock Detection and Resolution Scheme: Performance Study", in Proc Third International Conference on Distributed Computing Systems. 1982.496-501. Miami, FL

[JAGANNATHAN82b] Jagannathan J.R. and Vasudevan R., "Detection and Resolution of Deadlocks in Distributed Systems", Department of Computer Science, University of Calgary. Tech Report 82-108-27, Dec 1982.

[JORDAN] Jordan J.W. and Bruso S.A., "Improved Performance of Deadlock Avoidance Algorithms for Distributed Database Systems", The Mitre Corporation, Personal Correspondence.

[KAWAZU] Kawazu S., Susumu M., Menji I. and Kastumi T., "Two-Phase Deadlock Detection Algorithm in Distributed Databases", International Conference on Very Large Databases (VLDB) 1979 360-367.

[LOMET78a] Lomet D.B., "Multi-level Locking with Deadlock Avoidance", RC7019 IBM T.J. Watson Tech Report 3/8/78

[LOMET78b] Lomet D.B., "Coping with Deadlock in Distributed Systems", IBM T.J. Watson Research Center, RC7460. 12/29/78.

[LOMET80] Lomet D.B., "Deadlock Avoidance in Distributed Systems", IEEE Transactions on Software Engineering SE-6 No.3 March 1980.

[MAHMOUD76] Mahmoud S. and Riordon J.S., "Optimal Allocation of Resources in Distributed Information Networks", ACM Transactions on Database Systems, Vol.1, No.1, March 1976 66-78.

[MAHMOUD77] Mahmoud S. and Riordon J.S., "Software Controlled Acess to Distributed Databases, INFOR Vol. 15 No. 1, February 1977. 22-36.

[MARSLAND] Marsland A.T. and Isloor S.S., "Detection of Deadlocks in Distributed Database Systems", INFOR Vol. 18 No. 1 February 1980. 1-20.

[MENASCE78] Menasce D.A., Popek G.J. and Muntz P.R.," A Locking Protocol for Resource Coordination in Distributed Databases", ACM Transactions on Database Systems. (Also, International Conference on Management of Data (SIGMOD) 1978.)

[MENASCE79] Menasce D. and Muntz R.,"Locking and Deadlock Detection in Distributed Databases", IEEE Transactions on Software Engineering Vol. SE-5, May 1979. 195-202.

[MINOURA] Minoura T., "Deadlock Avoidance Revisited", Journal of the ACM, VOL. 29 No.4 October 1982. 1023-1048.

[MITCHELL] Mitchell D.P. and Merritt M.J., "A Distributed Algorithm for Deadlock Detection and Resolution", AT&T Bell Labs, Murray Hill, NJ 07974.

[MOSS] Moss J.E., "Nested Transactions: An Approach to Reliable Distributed Computing", Ph.D. Thesis. MIT/LCS/TR-260. April 1981.

[NATARAJAN] Natarajan N., "A Distributed Scheme for Detecting Communication Deadlock", IEEE Transaction on Software Engineering, Vol. SE-12 No.4, April 1986. 531-537.

[NEWTON] Newton G., "Deadlock Prevention, Detection, and Resolution: An Annotated Bibliography", Operating Systems Review, Vol. 13, No. 2, April 1979, 33-44.

[OBERMACK80a] Obermack R., "Global Deadlock Detection Algorithm", IBM Research Laboratory, RJ2845, 6/13/80.

[OBERMACK80b] Obermack R., "Deadlock Detection for all Classes", IBM Research Lab, RJ2955 10/6/80.

[OBERMACK82] Obermack R., "Distributed Deadlock Detection Algorithm", ACM Transactions on Database Systems Vol. 7 No. 2, June 1982, 187-208.

[Raeuchle] Raeuchle T. and Toueg S., "Exposure to Deadlock for Communicating Processes is Hard to Detect", TR 83-555, May 1983. Department of Computer Science, Cornell University.

[ROSENKRANTZ] Rosenkrantz D.J., Stearns R.E. and Lewis P.M., " System Level Concurrency Control for Distributed Database System", ACM Transactions on Database Systems Vol.3 No.2, June 1978.

178-198.

[SINHA84] Sinha M.K. and Natarajan N., "A Distributed Deadlock Detection Algorithm Based on Timestamps", IEEE Transactions on Software Engineering Vol.?? No.?? (Also, Fourth International Conference on Distributed Computing Systems, 1984 546-556.

[SINHA86] Sinha M.K., " Commutable Transactions and the Time-pad Synchronization Mechanisms for Distributed Systems", IEEE Transactions on Software Engineering, Vol. SE-12 No. 3, March 1986. 462-476.

[SOUNDARARAJAN] Soundararajan N.,"Axiomatic Semantics of Communicating Sequential Processes." ACM TOPLAS 6,4 (October 1984), 647-662.

[TSAI82a] Tsai W.-C., "Distributed Deadlock Detection in Distributed Database Systems. Ph.D. Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Jan 1982.

[TSAI82b] Tsai W.-C. and Belford G,, "Detecting Deadlock in Distributed Systems", INFOCOM 1982. 89-95

[TSAI86] Tsai W.-C. and Elmagarmid A.K., "Algorithms for the Detection and Resolution of Deadlocks in Distrbuted Database Systems" Submitted for Publication.

[WUU] Wuu G.T. and Bernstein A.J., "False Deadlock Detection in Distributed Systems", IEEE Transactions on Software Engineering Vol. SE-11, No. 8 August 1985. 820-821.

[ZOBEL] Zobel D., "The Deadlock Problem: A Classifying Bibliography." Operating Systems Review 17, 2 Oct 1983, 6-15