THE DEADLOCK PROBLEM IN COMPUTING AND COMMUNICATIONS SYSTEMS — AN ANNOTATED BIBLIOGRAPHY

by

Micha Hofri

Technical Report #500
March 1988

TECHNION — ISRAEL INSTITUTE OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE

HAIFA, ISRAEL
THE DEADLOCK PROBLEM IN COMPUTING AND
COMMUNICATIONS SYSTEMS - AN ANNOTATED BIBLIOGRAPHY

by

Micha Hofri

Technical Report #500
March 1988
THE DEADLOCK PROBLEM IN COMPUTING AND COMMUNICATIONS SYSTEMS  
—AN ANNOTATED BIBLIOGRAPHY

Micha Hofri  
Department of Computer Science, The Technion – IIT  
Haifa 32000 ISRAEL  
March 1, 1988


INTRODUCTION

This bibliography has been assembled during the Summer of 1987, while preparing for a students’ seminar on deadlock I organized here during the Fall term. It includes mainly entries from the leading journals and proceedings of the main conferences of Computer Science, with a few incursions from Electrical Engineering, particularly concerning communications systems. There are also a few technical reports listed – some of which I have not even seen, but have evidence to their worth, and that the results have not been published. I have also included a number of references dealing in the wider issues of concurrency control – and the list is far from exhaustive in this area.

Nearly every text on operating systems has some treatment of deadlock phenomena and handling. I have decided against listing those, as they are by necessity highly repetitive.

This is a preliminary release of the report. I request any reader who knows of worthwhile work which is not included, more accessible venues of listed items, corrections to bibliographic data, or wishes to add/change the capsule comments below—especially on his/her own papers—to communicate with me.

E-mail addresses: hofri@sel.technion.ac.il (csnet), hofri@techsel (bitnet).


A centralized algorithm to check whether a store-and-forward network is in buffer deadlock. The author mentions several nontrivial restrictions on the environment, so that the applicability of the procedure is not clear.


Considers a "routed network" with flow-control: routes between pairs of nodes are fixed (a pair may have more than one route defined for it), and every route is only allowed to keep up to a given upper bound of unacknowledged messages. The proposed algorithms have a flavour similar to those in the work of Kameda (1980), but appear to be somewhat earlier.

An elaboration of the Banker's Algorithm, with an additional process that maintains lists of allowed allocations in advance, to shorten the actual allocations delays. Well written; but requires a serious average time estimate, and not just worst-time complexity.

Is this an improvement on Fontao's (1971) approach?


Centralized algorithm: T resource types, \( t_j \) of type \( j \), \( ALLOC-FREE \) logical flow of all processes given, with no loops but conditional branches allowed. Avoidance is shown to be available in P cost when \( ALLOC-FREE \) pairs are nested, and each pair is for a single resource type. NP-completeness is proved for a few other cases.


A procedure to convert the distributed dynamic resolution with local information problem to a static one with complete advance information on requests. The procedure consumes time and communications costs inferior to those that need be expended to resolve the static one. No proofs. Very neat.


A Turing award lecture on the transition from processor centered computing to data centered one, into which he inserts some observations from experience in developing DBMSs, including a popular one (IDS) where deadlock is treated by scotching the guilty transactions, and 10% of all traffic is thus aborted!


An algorithm that seems to have been inspired by Obermark's (1982), but claims to be an improvement, in that it saves some inter-site messages, by making a distinction between two types of resources, depending on whether the remote resource lock granularity and mode (shared or exclusive) – when it is locked – can be determined without accessing the remote site. An algorithm is presented, explained and "informally proved". Some consideration is given to its performance, starting from the observation that most deadlocks are two-site deadlocks, sometimes three sites are involved, and rarely more.


Appears to have been superseded by Badal (1986).


Experimentally (mainly simulations) based, low-keyed observations on the behavior of a small distributed transaction system under various concurrency cum deadlock control protocols. The main result: locking that leads to deadlock, and consequent restarts may result in a thrashing phenomenon very similar to the one observed in VS systems when storage is over-committed. Comparisons of several such protocols show intuition to be a fallible predictor of their relative efficacy.


Three types of resources are considered: Locks, M out of N and pool resources (such as variable size storage
allocations). The detection algorithm does not process a complete resource graph, but maintains the information in essentially equivalent lists. Candidates for the resolution of a detected deadlock are proposed by the detector, and a resolution algorithm based on this specification is presented. The possibility of this operating in a distributed environment is mentioned. Careful proofs, but hard to follow. An interesting performance discussion is provided.


A report on extensive simulation, providing also information on the incidence of deadlocks among transactions as a function of the concurrency level.


Models that are based on wait-for graphs are presented for deadlock detection and reasoning about properties of deadlocked systems. The usual wait-for graph is augmented so that the "wait relation" is allowed to be formed, in addition to the obvious "AND", also with the "OR" operation. (This allows requests for multiple-unit resources to be represented in a wait-for-graph, but presumably other interesting possibilities exist as well).


A comprehensive, brief survey (mainly citing references) of types of deadlock in store-and-forward networks and their remedies. Two new protocols for avoiding the standard buffer deadlock are proposed.


Presents simple-minded procedures that are demonstrably optimal in the number of buffers they need in order to function correctly. These procedures are also shown to have properties that are expected to be rather undesirable in actual use. Well written.


A detailed presentation of a simple—and often recurring—idea that avoids buffer deadlock in store-and-forward networks: keep aside "exchange buffers" and use them only when in trouble, to locally resolve the congestion.


An elaboration of procedures presented in another paper by the authors in the same issue, where only buffer deadlock was discussed. Entirely sensible and straightforward.


Chapters 8 (pp. 227-293) and 9 (pp. 295-350) consider respectively centralized and distributed computer systems; deadlock management is not the central topic, but forms the main constraining attribute. In Chapter 8 the emphasis is on the deadlock-avoidance work of Cellary, and in Chapter 9 on the work of Blazewicz on deadlock in Store-and-Forward networks.

All routing is done over an acyclic graph embedded in the network. Over this restricted set of routes a minimal assignment of buffers is feasible. Comparisons with other buffer allocation schemes are given.


Processes issue requests in the form "m-out-of-n" (from a specified set of processes of size n). Exclusive requests, and common OR requests are natural simple cases. The set-up is general in the sense that the processes may request different types of events, like process message or termination. Such events are bound to be produced by the requested processes if they are not blocked. The events may represent either single unit resources or actual messages (consumable entities). It is shown that by introducing "dummy" processes a rather complicated variety of requests may be described under the constraint that a process may have only one outstanding (general) request. Deadlock detection algorithms are given first for a static system, and then this is expanded in two stages to a system that may change its state during the detection activity. Proofs are often omitted (a TR is referenced).


Describes a technique to convert a system of concurrent processes (with process creation and deletion) to a Petri net, and analysis that checks whether a deadlock is possible. The analysis seems routine, and not specific to the distributed environment.


A model that applies to a fixed set of otherwise arbitrary set of concurrent processes, each consisting of a straight line sequence of arbitrarily ordered semaphore operations. Any interleaving of of such operations defines a "progress path" for the system, which has to keep out of "unsafe zones" if execution to completion is to be guaranteed. The 'straight-line' restriction means that the path may not double on itself. Static deadlock detection. Also partial deadlock(?)


Considers a t-unit single-resource system, where all processes have known maximum requirements, and the purpose of the paper is to demonstrate a Banker-algorithm-type safety test that is computationally simple (a single comparison). The price is relatively many changes in the basic data structure on each allocation or release. Very nicely done.


Appears to be a straightforward implementation of Cellary (1979) to limit initial allocations of resources. The presentation is confusing (better in Blazewicz et al. *Ann. of Oper. Res.* 7, 1986?)

(26) Donald C. Chamberlin, Raymond F. Boyce, Irving L. Traiger: A Deadlock-Free Scheme for Resource Locking in a Database Environment. In *Information Processing 74*. Amsterdam, the Netherlands: North-
Hofri: Deadlock Bibliography


An astute discussion why "standard" deadlock avoidance methods won't do for a database environment. Then protocols are presented for record locking and unlocking:

1) Locks must be set in one SEIZE block, inside which no processing of the records is performed, and after their processing is completed (presumed consistent) they must be all released in one RELEASE block before any other locks may be SEIZED. This will guarantee that no data in an inconsistent state need be preempted, since processes may only be blocked before they are allowed to update any data. (Essentially 2PL.)

2) An algorithm is then proposed and proven (proof not examined) to implement the processing of such SEIZE blocks avoiding deadlocks.


The algorithm is described in Goldman (1977), and failure modes for it are shown (it is a simple table concentration procedure (for each node in the network)). The authors realize that the overhead in table transmission could be unsupportable, and attempt to save in table transmissions. It is shown in Goldman (1977) that the adopted protocol may lead to overlooking certain deadlock configurations.


Defines a "system" as a collection of processes which only interact by messages over predefined links, à la CSP. Defines possible states of these links and processes, and deadlock in terms of these concepts. Then shows that deadlock-freedom requires the existence of a "proper" set of priorities over all edges. Checking for, or proving the possibility of existence of such a set is claimed to be easy and thus usable as a tool for reasoning about existence and liability of deadlock. Very terse.


A clearly written distributed deadlock detection protocol over wait-for graphs, both for single- and multiple-request systems.


Generalizes the dining problem in that the great minds may be related via a graph more complex than a cycle; here too each edge carries a resource (a bottle) but drinkers may crave at different times different subsets of the resources available to them (to mix different cocktails, presumably). Only tangentially relevant to deadlock management.


Considers both communication and resource deadlocks. The first type is more general, but resource deadlocks (where a process needs all the resources for which it has issued requests to be allocated before it can proceed) can be detected by a simpler algorithm, so they are treated separately. Detection of all existing deadlocks is assured, with no phantom deadlocks reported. No global information exists. Communication is only by uniform short messages, which are transmitted in FIFO order, but their delays are unbounded (though known to be finite – the Dijkstra and Scholten model). Algorithms are given and proved, and their performance issues are briefly discussed.
(33) Marina C. Chen, Martin Rem: Deadlock Freedom in Resource Contentions. Acta Inf. 21, pp. 585-598 (1985). Under the expedient resource allocation policy, two necessary and sufficient conditions for deadlock are demonstrated, and are shown to be equivalent. (One was discovered in Ibaraki and Kameda, 1982). The conditions are related to the König-Hall theorem on the existence of a system of distinct representatives for a collection of sets.


(35) Israel Cidon, Jeffrey M. Jaffe, Moshe Sidi: Local Distributed Deadlock Detection by Cycle Detection and Clustering. IEEE Trans. Soft. Eng. SE-13:1, pp. 3-14, (1987). A Store-and-Forward communication network is considered. The detection is initiated by any node once the portion of its buffer space that is allocated for normal traffic fills up. The authors compare the basic ideas in their algorithm with those of Misra and Chandy (1982), Bracha and Toueg (1984) and Mitchell and Meritt (1984). They describe their procedure as a modification of the last two. The procedure consists of detecting cycles of full nodes (called clusters here), and consolidating them into knots. The notion of "tie" (a knot plus whoever that can reach it) is introduced as well. A "static" algorithm is described, during which no node changes its status. The static procedure is used as a building block in a "dynamic" version, where nodes (that can do so) continue their communication activity. The algorithm operates in a storage kept for the purpose in each node; its size is proportional to the number of links incident on the node.


(38) Ludwik Czaja: Deadlock and Fairness in Parallel Schemes: A Set-Theoretic Characterization and Decision Problems. IPL 10:4,5, pp. 234-239, (1980). A formal setup is defined where the questions of deadlock and starvation (and its converse: weak and strong fairness) can be discussed. Several such problems are phrased. No algorithmic discussion.

(39) Ajoy Datta, S. Ghosh: Synthesis of a Class of Deadlock-Free Petri Nets. JACM 31:3, pp. 486-506, (1984). On rules that only produce nets that belong to a set called regular nets, for which liveness and safety properties can be trivially assured. There is no indication how versatile this class is, as a descriptive tool, but the authors claim it is adequate for many design tasks.

(40) R. Devillers: Game Interpretation of the Deadlock Avoidance Problem. CACM 20:10, pp. 741-745, (1977). Presents a framework suitable for reasoning about properties of systems of processes when their logical flow is known (with branches and loops allowed). Reasonable definitions are given for safe, unsafe and doomed states, and an algorithm is presented to construct the sets of such states. The motivation for the approach cited, i.e. the discrediting of the "Maximum Claims Known" approach. The author claims the main benefit of the method is the provision of tools of reasoning to the designer of such systems, but little evidence is provided, beyond reference to his thesis, of which this paper is a summary.


A counting argument, for $m$ processes over $r$ units of one resource, showing that the probability of reaching deadlock increases with $m$. Rather too simplistic.


A classified and annotated survey.

Presemts a brief classification of deadlock detection in distributed systems and proposes a new one. Though called distributed, it is so only in part. The idea is that a system-wide representation of the Wait-for graph is kept in disjoint pieces in various sites, each piece containing transactions that are known to be in interaction over resource(s). (Apparently transactions reside at a single site, so that the resource status of the transactions that appear in that portion of the graph is locally available). Status changes of resources and transactions are incorporated locally into the resident graph-portion; certain changes are deemed deadlock-prone (the natural ones) and will trigger a complete deadlock detection procedure. This one is proposed to be handled by selecting a controller (one of the sites, chosen by turns) and forwarding to it from the entire system the pieces of the Wait-for graph so it can join them and proceed via a usual cycle detection algorithm. No proofs provided.

Consistency and avoidance of conflicts in database systems are discussed from first principles. Easy implementation of transaction backup and recovery are also cited as reasons for locking-based operation of such systems. Two-phase-locking is argued to result naturally from the requirement of consistency. Predicate locks are described as a help in situations where the set of entities that should be locked is not known (not even known to exist) a-priori, and their management is outlined via examples. The treatment is entirely non-mathematical and well written. Deadlock is not addressed explicitly at all.

Presents a generalization of sorts of the "Banker's protocol", which could be of interest in a multiprocessng system. Each process submits a claim vector (called here "want" vector). A scheduler maintains a matrix $R$, the $(i,j)$th element of which contains the number of units which process $i$ may be allocated of resource $j$, without leaving the set of safe states. An efficient algorithm is shown and proved for the computation of $R$. The design of a scheduler around this data structure is discussed in some detail. The avoidance of certain types of "effective deadlock" is discussed as well.

Provides a detailed treatment of the resource allocation policies used in the MACE operating system,
developed at Purdue University around 1970 for CDC-6000-based systems. The design goals of the resource managers are clearly enunciated and their relative costs and merits are discussed in the context of the description of the basic protocols. The important decision made was to treat most SR resources as "preemptive" (preemptible) most of their life time. No mathematics, nothing explicitly quantitative – though protocol execution costs seem to have ranked high in the design process.


A theory on the use of a-priori syntactic information about the behavior of the transactions, and how it can be used to construct correct locking protocols, assuring the preservation of data consistency. The relationship between assuring correctness and deadlock-freedom is explored in a unified model which allows both shared and exclusive locks. Very smooth.


A general framework for concurrency control, where resources are requested for specified operations; the definitions of operations state which operations are mutually compatible (require no exclusive control). Furthermore, such operations may be defined on an object-by-object basis. These definitions allow the system resource managers to form the required Wait-for graph for any combination of ownership and pending requests. Deadlock detection and deadlock avoidance are then discussed using these graphs in standards terms (detection is via recursive reduction by elimination of sinks, and avoidance by performing same over a "claim graph", with some modification of the data structures to improve efficiency).


Presents a seemingly novel organization of the message buffers in Store-and-Forward packet switching network, that avoids most of the costs and hassles of the earlier schemes, and avoids buffer deadlocks. This is done at the cost of rerouting packets to avoid congestion zones, and reserving a large buffer array in one distinguished node, which may resolve emergencies by losing (overwriting) packets. The algorithm is well specified and the proof appears good.


Appears to be a variation on Havender's scheme; wrong as presented, and the components do not hang well together.


A "negative" paper: the deadlock detection protocol of Menascé and Muntz (1979) is shown by a counter-example to ignore some of the information required to detect multi-site deadlocks. A remedy suggested by Menascé is illustrated. It takes care of the counter-example, and the need of a proof for the improved procedure is pre-empted by explaining that the resultant protocol is unattractive, due to inherent inefficiencies.
This type of inefficiency is claimed to beset other similar protocols as well.

The discussion is within the framework of the Banker's Algorithm, but introduces the concept of *partial requests*, which have the form "if I am not allowed now to have my full requirement $c$, I could use a smaller allocation $y$, and after using it for some time release resources (possibly only part of my holdings), until my needs can be satisfied in full". It is shown that when in addition to the claim vectors $c$ also an arbitrary collection of partial requests is presented to the Banker, the problem of deciding whether allocations are safe is NP-complete (by reduction to satisfiability). Then it is shown that under some regularity constraints on the possible partial requests there exist decision procedures to determine safety that are of acceptable computational complexity.

(Ph.D. thesis) Describes and proves a deadlock avoidance—the title is misleading there—protocol (both centralized and distributed) that maintain an Ordered Blocked Processes List in each node. The distributed protocol is well proven, but appears quite profligate in its communications requirements. Demonstrates failures of two published algorithms (Chandra, Howe and Karp (1974), and Mahmoud and Riordon (1976)).

Under assumptions of symmetry and independence, with millions of lockable entities ($N$) and hundreds of transactions, each one locking up to a few hundreds of entities—those probabilities are minute, and linear in $N$. More careful computations were done by Massey (1986), and also (without considering the deadlock phenomenon directly) by D. Mitra and P.J. Weinberger (1984) and D. Mitra (1985).

Comparable with Merlin and Schweitzer (1980), and bitter.

Authors claim this is derived from the 1982 Chandy & Misra deadlock detection algorithm, with better performance characteristics, and allows addressing the question of resolution as well. No phantom deadlocks are found.


A leisurely account of deadlock avoidance when information on maximum claims is available, first with one resource type, and then with several. The approach is similar to Habermann (1975). Well written. Complexity issues are only briefly mentioned, but motivate some of the presentation.

The repertoire of statements within CSP is presented, and (distributed) programs are defined as CSP statements. "Matching" commands are defined in terms of the usual communication primitives, and thus obtains the notion of a deadlocked program execution, as a sequence of "configurations", which ends with a configuration beyond which it cannot continue. This construct can display what the authors call "inherent deadlock" – essentially, either an infinite loop or an unmatchable IO statement. A canonical representation is defined for a program, and in its terms a necessary and sufficient condition for the existence of an inherent deadlock is given. Several other (partial) conditions that give rise to more efficient algorithms are discussed as well.


As the date requires, the paper presents an introduction to the deadlock phenomenon in computing systems, and then serialized allocation has its first presentation. The paper is written in terms of a particular operating system – the IBM/360 OS(MVT). The resource allocation done by the job initiators is detailed. Of the three resource classes – files, main storage and devices, the files are allocated first, for the entire job duration. The others are allocated for a job-step at a time, and there is a demonstration (dependent on the internal flow of control in the inititator) why storage has to be allocated before devices (both are posted as free at end of a job-step, but the inititator tries to avoid actually changing these allocations from one step to the next). There is an interesting discussion showing why files have to be allocated for the entire job-duration, rather than per job-step.


A true generalization of Habermann's Banker algorithm. The presentation, however, is not complete, and relies for much of its effect on his thesis (Deadlock-Free Sharing of Resources in Asynchronous Systems, Project MAC TR-75, September 1970).


A more complete version is available as INRIA RR #493 (August 1986). The system and communication model is apparently the same as that stipulated in Chandy, Misra and Haas (1983). The authors' idea is to make do with fewer messages by appending to each message the list of processes that are its intended recipients (or have contributed to it, when it is a response message). The messages can become thus relatively large; hopefully the saving in their number is dominating.

(71) T. Herman, K. Mani Chandy: A Distributed Procedure to Detect AND/OR Deadlock. *AFOSR-TR-83-0989* (LCS-8301, Department of Computer Science, the University of Texas at Austin) (Jan. 1983).


Present a few message-passing based algorithms, that give rise to different numbers of required messages, to construct a consistent wait-for graph. Subsequently the graph is searched for a cycle. Transactions that do not follow a 2PL scheme may cause the reporting of phantom deadlocks.

Hofri: Deadlock Bibliography

An extensive listing (more than 150 entries), mainly works that address deadlock issues explicitly, and some on theoretical issues of concurrency control. It omits textbook treatments of the subject, and gives informal capsule comments for those entries the compiler has read.

A rather complete discussion of effective deadlocks. See Parnas and Habermann (1972).


A rather straight-forward application of Kameda (1980): classes of processes are given, and the problem is to determine the number of processes from each class that can be included while still maintaining the conditions that are shown in the above reference to be necessary and sufficient for deadlock freedom. Most of the paper is given to the discussion of the worst-case complexity of the required tests.

A rather elaborate and heavy-going use of the results of Kameda (1980) (The results in Ibaraki and Kameda (1982) are used to some extent as well). The first fact shown is that when the system has n processes a total "deficiency" (difference between maximum claims, which are assumed known, and the given supply of resource units) of n−1 units still allows deadlock freedom. Each resource is associated with a cost per unit, and the paper suggests an efficient algorithm to parcel the allowed deficiency over those resources.

The authors present their approach as an improvement of Goldman's (1977) algorithm in two senses:
a) Processes are allowed to have more than one outstanding request.
b) Deadlock detection is not carried out periodically, but maintained continuously.
The idea is to maintain at each node, an up-to-date version of the complete resource graph, so that there is no need for extensive transmission of status information when deadlock detection is decided upon; rather, each allocation decision is immediately diffused throughout the entire system. Algorithms are given for the various updates that are required. They are not proved, though the authors are aware of the pitfalls of synchronization. (It is not clear yet if they are all correct). Quaintly enough, although the resources are inherently "single unit resources", they are kept fully represented in the resource graph, rather than shrink it to a wait-for-graph.

(84) Jeffrey M. Jaffe, Moshe Sidi: Distributed Deadlock Resolution. IBM RC 12541 (#56407) March 2 1987.
Propose the often-used idea of resolving store-and-forward deadlocks by reserving a fixed amount of storage in each node in the network for rescue operations. The unusual feature of the proposed algorithm is that there is no attempt to actually identify a deadlock state. What is done is to detect congested regions, where nodes may either be already deadlocked, or are likely to get soon involved in one, and relieve the congestion "locally".

Proposes an extension of the deadlock detection protocol of Goldman (1977), to allow multiple resource requests (and partial allocation). The extension is not proved. Resolution is briefly and nonconclusively discussed. A simulation experiment is reported that shows the benefit of the above extension in a small system.

Concerning the algorithms proposed in Ho and Ramamoorty (1982), showing possibilities of phantom deadlocks.

The paper states as its motivation the fact that detecting a deadlock is equivalent to computing the transitive closure of a directed graph (such as a wait-for graph. The discovery of a loop, if one exists, is expected to be a subsidiary result of any algorithm that computes the closure). They then consider a particular, simple, algorithm analyzed by Schnorr for its expected execution time, under assumptions of uniform distribution of a given number of edges over the graph and independence of the adjacency lists of distinct nodes. They show that since when a graph is known to be acyclic the adjacency lists are not independent, the estimates of Schnorr can lead to substantial underestimates of the running time of a deadlock detection algorithm.

Good, short, well-written survey of complexity results pertaining to deadlock. Shows basically that any interesting prediction one may want to make about deadlock is NP-hard.

A leisurely, well-explained procedure. Considers a known set of processes, with their maximum claim matrices and a fixed supply of SR resources managed according to the expedient allocation policy (=immediate allocations). The problem of testing for safety is reduced to a node-capacity-constrained matching over a bipartite graph, which is in turn reduced to a specialized maximum-flow problem.

Presents distributed algorithms for the modification of directed graphs without creating cycles. Asynchronous environment, numerous overlapping (spatially and temporally) instances of the algorithm are allowed. The applications to on-the-fly deadlock avoidance are discussed.

(A two-phase protocol means here that local and global phases alternate. However ~ ) Elmagarmid reports
that this protocol has been shown to be incorrect, producing phantom deadlocks, and when more than one resource is requested, can lead to overlooking global deadlocks as well.


The paper proposes an implementation of LOCKing mechanism for a system with single-unit resources, and analyzes an approach to deadlock detection and resolution on a graph which involves processes and resources, but differs from the Holt resource graph: Each process points at the first resource it acquired, from which an arc extends to the next one allocated to it etc. Since the outdegree of each node is one, a deadlock state requires (and is generated by) a cycle. They suggest checking whenever a request is blocked. There is some nonconclusive discussion of resolution.


An improvement of Habermann’s version of the Banker’s algorithm, with lower time complexity ($O(n+m)$ instead of $O(nm)$).


System/R-like locking protocols are defined and extended to assure deadlock-freedom. No performance issues.


IBM T.J. Watson Research Center Report(?)

A well written prescription for the development of a distributed algorithm that must operate in a dynamic environment, from a centralized, static one. Deadlock detection is brought in as an example; the algorithms of Goldman (1977) and Haas (1981?) are treated through this process. The authors than show that their method of synthesis allows them to combine fragments of different algorithms, and propose a new, hierarchical distributed deadlock detection algorithm.


The process scheduler of the OS Boss 2 (for the RC 4000 machine) is described. The designers were concerned about deadlock, either real or effective. The stated goals are “to provide reasonable turn-around time and avoid deadly embrace”. Deadlock is addressed no further except than to note that the priority updating mechanism provides for any request (not exceeding resources) being satisfied within a finite time.


Builds upon Yannakakis, Papadimitriou, Kung (1981) to insert Locks and Unlocks in transactions such that the resulting set is safe and deadlock-free. Differs essentially from 2PL; there are cases where each allows higher concurrency than the other. The algorithms and their correctness proofs are set in a suggestive geometrical landscape, (the same one used in Eljas Soisalon-Soininen and Deric Wood (1982)), created by using the actions of the transactions to form a multi-dimensional grid (nearly all the required mathematics are
done on two-dimensional projections, though). Wait-for graphs are mentioned, but not really used. The description of the algorithm is rather verbose.


Centralized system: Given the number of resources, unit count of each, a set of deadlocked processes with their request and allocation vectors and the cost of preemption each of these processes – discusses the problem of finding the subset that minimizes the sum of preemption costs. Proves NP-completeness. Produces three heuristics, bounds their computing time and gives some simulation experiments.


Intricate algorithm which builds upon Yannakakis et al. (1979) to determine deadlock freedom of 2 transactions in $O(n \log n \log \log n)$. Note that $O(n^2)$ is obvious.


On how to resolve a deadlock caused by a new request by a single processor via a particular network flow problem. (Compare with Ibaraki & Kameda?)


When only single-unit resources need be considered, Haberman’s (and Holt’s) resource graph is adequately represented by a wait-for graph. This fact is extended to the claim-graph described by Haberman; a necessary and sufficient condition for safety in such a system is the acyclicity of this graph. A deadlock avoidance procedure that amounts to checking each allocation for cycle formation is described. Allocations can be in a shared or exclusive mode.


Suggests a non-trivial notion of partitioning a task-resource system so that implementing safety checks can be simplified. Worst case behavior is apparently not affected (possibly adversely), but it appears that normal execution sequences could gain some. Also a resolution of “effective deadlocks” (indefinite delays) is presented in this terminology. The overall merit remains to be demonstrated. The discussion is in terms of single-unit resources, but this limitation appears inessential.


A distributed collection of processes is arbitrarily organized in a hierarchy, and an extended Banker’s Algorithm operates with moderate amount of interaction. Measurement of performance is reported.


The deadlock issue is not considered explicitly.


Two deadlock avoidance algorithms are described, with the common relation: a centralized one and a related distributed version. The centralized is entirely equivalent to cycle detection algorithm in an isolated system
with a central allocator; there appears to be no problem with this protocol. There is however also a
 decentralized avoidance procedure, but it can allow a deadlock go undetected. Goldman (1977) describes the
deadlock avoidance algorithms proposed in this paper and gives an example for the above possibility of error.

The authors argue for deadlock detection on-the-fly, rather than periodic analysis, which calls for much
 disruptive communication. Fairly detailed description of the state of knowledge at 1978. They propose such
 an algorithm that only communicates information about "global" processes, keeps for each its reachability set
 on a Wait-for graph, on a continuous basis. They provide examples, and no proofs (though aware of the
 fallibility of such protocols).


(111) Daniel A. Menascé, Richard Muntz: Locking and Deadlock Detection in Distributed Data Bases. *IEEE

355-360.
A detailed account of possible types of deadlock that can arise in such networks over the buffers allocated for
packets, either in transit or at the destination nodes. Several remedies are provided and proved in detail. The
solutions involve a graph over the entire set of buffers in the network. Minimizing the number of buffers
required to assure deadlock-freedom is the main criterion used to evaluate protocols.

Algorithms over a Wait-For graph are defined (allowing single unit resources only). Several of the algorithms
appear to be new; the issues of concurrency level, complexity of the detection, test for deadlock-freedom and
deadlock-avoidance are discussed.

(1982).
A simplification of similar work of Dijkstra and Scholten. Each node corresponds to a process. An edge in
the graph corresponds to a full duplex channel and defines the "neighbor" relation. Each node only knows his
neighbors, and is ignorant of the rest of the network. The communication subnetwork is assumed never to
mutilate or lose a message, and deliver them in FIFO order between neighbors. Each node has to maintain a
buffer that can hold as many messages as it has neighbors. The authors show how the same approach can be
used to agree on the termination of a cooperative computation.

Two simple algorithms over a Wait-For graph. A process detecting it is in a deadlock aborts, thus providing
myopic resolution.

(116) Debasis Mitra and P. J. Weinberger: Probabilistic Models of Database Locking: Solutions, Computational

(117) Debasis Mitra: Probabilistic Models and Asymptotic Results for Concurrent Processing with Exclusive and
A detailed and comprehensive discussion of task scheduling when static priorities provide a total order over the processes. Several allocation policies are described, and their vulnerability to deadlock and effective deadlock is discussed, with nice, but very small scale examples; there is little analysis beyond pointing out particular phenomena that occur in the examples. The authors in particular concentrate on the relation between the two orderings of the tasks: that of the priority and that of the arrival (or start) times, when selection for allocation of a resource among several waiting tasks must be made. They exhibit algorithms for this purpose (written in Algol). Their main properties are only informally proved.

An early paper; explains the deadlock problem from scratch. Gives a detailed description of an algorithm for the management of a request queue by a resource manager, which incorporates a deadlock detection protocol. The resources are all single-unit ones, and requests can be for Exclusive or Shared allocation. The detection consists essentially of looking for a cycle in a structure equivalent to a Wait-for graph.

An O(1) storage, "on the fly" communication deadlock detection algorithm suitable for dynamically evolving environments, that does not require "freezing" the system.


Appears to be one of the first "path-pushing" (Bernstein, Hadzilacos and Goodman terminology) algorithms. Used in the best documented Distributed Database System, R*. May "detect" false deadlocks.

On deadlocks (or rather, livelocks) that arise because of transmission errors, and interesting algorithms that assure recovery.

Clarifications for the concepts 'safe state' and 'permanent blocking' (=effective deadlock). No surprises.

Discusses, among other issues, the deadlock management of the MTS system.

Deadlock is not treated on its own. The emphasis is on flow control, and deadlock is handled by the usual scheme of numbering buffer classes.

On the freedom from deadlock of communications protocols via global state analysis, whether the channels are FIFO or not, bounded or not.

(129) D.J. Rosenkrantz, R.E. Stearns, P.M. Lewis: System Level Concurrency Control for Distributed Database Systems. *ACM Trans. on Database Systems*, 3:2, pp. 178-198 (1978). Deadlock is treated as a nuisance result of the concurrency control which is imposed for consistency. It is effective deadlock, rather than complete deadlock that is emphasized, as likelier to occur in an environment which aborts and restarts processes. To avoid such a possibility the authors describe priority-based deadlock avoidance schemes called Wound-Wait and Wound-Die. The protocols are proven correct; their impact on system performance is only informally compared.

Consider deadlock detection for single-unit resources which may be accessed in either shared or exclusive mode. Then an algebra of access modes is developed, to discuss deadlock avoidance. The authors claim this algebra is useful e.g. for the CODASYL scheme, where parts of a resource that is locked in its entirety may be changed independently. This discussion is not clear to me. Reports on a minor error in the King & Collmeyer (1973) paper.

A database with exclusive and nonexclusive locks is considered, in a central environment, where processes issue a single request at a time, and each request is tested for creating a deadlock. The required data structures are considered, with the operations specified at a fairly detailed level, but with no estimates of the involved overhead.

Describes an implementation of a DBMS employing Havender's scheme, with minimal variations, to facilitate accesses in a non-predetermined order.

On maintaining a dynamic DG as a DAG, by doing an abbreviated DFS prior to each edge addition. Useful for Wait-for graph maintenance.

The paper reflects its period in discussing from scratch the danger inherent in such an environment. Discusses shared and exclusive usage, describes "walking protocols" (that lock data as the need arises, and release as soon as consistency allows) and points them out as efficient and deadlock-prone.


On a characterization of deadlock in transition systems, with particular consideration for Condition-Action (CA) and Petri net schemes.

Develop a family of non-2PL locking protocols that assure serializability and deadlock-freedom. This family is claimed to encompass all previously formulated non-2PL protocols. Performance/concurrency levels are not mentioned.


The authors use the same geometrical model of locked transactions as Lipski and Papadimitriou (1981) and show how it can be used to test for safety of specified schedules. They manage however to drop an loglog n factor from the runtime of the algorithm proposed by Lipski and Papadimitriou.


Detailed presentation of algorithms for deadlock detection (in local and global phases) and deadlock resolution, with proofs. Performance analysis is given as well, but appears to rely much on uniformly random properties of the underlying Wait-for graph. (The proofs have still to be examined...).


In error (Shown by Wolfson and Yannakakis (1986).)


A Store-and-Forward packet switching network with fixed routing (source-sink dependent) and local transmission controllers is considered. Observes that common protocols that avoid deadlock may allow livelock (=effective deadlock: some packets may be indefinitely discriminated against and kept immovable) under excessive load on some of the routes. A modification of a protocol that was shown in Toueg and Ullman (1981) to be deadlock-free, so as to consider also the generation times of packets that are waiting to enter a node (these times are considered part of the local state of that node) is shown to be livelock-free as well.


A unified model of concurrency control algorithms for distributed databases, can be used to model 2PL, timestamp-ordering and others; a new algorithm, free from deadlocks and transaction restarts is shown. Different transactions can be executed under different concurrency controls simultaneously.


Extends Havender scheme to distributed transactions; provides an efficient sufficient test for deadlock.
freedom; discusses transaction copies.


Shows that testing 2 distributed transactions for deadlock is NP-complete (contrary to the result claimed in Tirri 1983), but testing them for safety and deadlock freedom is polynomial.


Journal versions of Yannakakis et al. (1979). The most important result from the deadlock point of view is that testing an arbitrary set of centralized transactions for deadlock freedom is NP-complete even if the transactions are nice and use Two-phase-locking.


Treats deadlock and serializability in a unified, geometric elegant model.


A polynomial protocol is described, to check whether two FSMs which use a single type of message—equivalent to saying that all 'send's and 'receive's are identical—may reach deadlock; (the running time may be cubic in the product of the number of states.)


An elaboration of Hao and Yeh (1982); abstracts the basic constructs of CSP that are relevant to the possibility of deadlock, and defines "inherent deadlock". An algorithm for the detection in a CSP program the capability to enter such a state is given.


A formalization of the work presented by the same authors in COMPSAC 1984, that allows for a mechanical determination of the necessary "discriminating equations".


Describes a procedure based on two earlier papers of the same authors, using terminology that is specific to ADA.