Efficient Techniques for Deadlock Resolution in Distributed Systems

Yuan-Chieh Chow  William F. Klostermeyer  Kenneth Luo

Computer and Information Sciences Department
University of Florida
Gainesville, FLORIDA 32611

Abstract

Resolving a deadlock in a computer system involves choosing processes to terminate or roll back so that the deadlock is eliminated. It is widely known that the problem of finding the minimum number of processes to abort is NP-complete. Two polynomial-time heuristics to resolve deadlocks, the edge cycle method and a method based on enumerating the cycles in the wait-for-graph are discussed. Simulation results are presented that show these heuristics perform better than several previously reported heuristics.

1 Introduction

Deadlock occurs in multiprocessor and distributed computer systems when two (or more) processes are waiting for each other to release some resources. This is commonly called a circular wait [1][2] and causes all of the processes involved to be blocked forever, or until the deadlock is resolved. Resolution involves aborting some subset of the processes so that the remaining processes can proceed.

There are several processing scenarios in which deadlock can occur. Independent processes in distributed and multiprogrammed systems may share hardware resources such as disk drives and printers that can result in circular waits. Processes on a parallel programming machine implementing shared variables can suffer from deadlock if processes are contending for common shared variables. Distributed databases suffer from deadlocks if multiple transactions use a common set of records of a shared database.

The resource request and allocation state of a system can be modeled as a directed graph called a wait-for-graph (WFG) [1][3]. Deadlock is present in the system if the WFG contains cycles. The problem of optimally resolving deadlock is reduced to finding the smallest subset of the vertices in the WFG that can be removed so that the WFG is cycle-free.

Let \( G=(V, E) \) be a WFG with \( n \) vertices and \( m \) edges. Furthermore, assume that \( G \) is connected (otherwise consider the connected components of \( G \)). An abort set, or feedback vertex set, \( S \), is a set of vertices \( S \) such that the induced graph \( G=(V-S, E) \) is acyclic. However, finding the minimum feedback vertex set is NP-complete [4]. As it is costly to stop a process, roll it back (or terminate it), reclaim its resources, and restart it, the system software must employ some heuristic to efficiently resolve deadlock.

We assume that a computer system employs one of the well-known deadlock detection algorithms, such as those in [1] and [2] for multiprocessor systems or those surveyed in [6] for distributed systems, many of which are based on the algorithm of Chandra, Misra, and Haas [7]. We present two algorithms that effectively resolve deadlocks in polynomial time. Our algorithms take as input a WFG constructed by a deadlock detection algorithm.

2 Related Work

The simplest method of deadlock resolution, given in Mitchell and Merritt [8] and in Roesler and Burkhard [9] is for any process that discovers that it is

This research supported by Florida High Technology and Industry Council
involved in a deadlock to abort itself. Figure 1 shows how the "suicide" approach can, for a WFG with \( n+4 \) nodes can result in \( n \) processes being aborted (for any \( n \)) when the optimal abort set consists of one node, i.e. node \( n+4 \). This implies the abort ratio to the optimal is \( n/l \), which is undesirable. In Figure 1, each process \( i \) determines that it is deadlocked at time \( i \). Note that there are edges from vertex \( n+4 \) to all vertices 1 to \( n \) and edges from vertex \( k \) to vertex \( k+1 \) for \( k=1 \) to \( n-1 \).

![Figure 1. Suicide Resolution is Not Optimal](image)

A similar approach is taken by Elmagarmid et al. [10] for distributed database systems. When some transaction manager discovers that a transaction is deadlocked, the resources allocated to that transaction that are causing the deadlock are released.

Sinha and Natarajan [11] resolve deadlock by aborting the lowest priority member of the cycle. Like the suicide approach, this does not consider the structure of the WFG, and can lead to a large number of unnecessary aborts: the number of processes aborted may be \( O(n) \) times greater than the minimum number.

Awerbuch and Micali [12] discuss resolution in detail, however their focus is on the correct, efficient construction of a resolution structure. Their algorithm results in a "minimal" set of aborts, minimal meaning no process is aborted that is not involved in the deadlock. While it is important that no such process be aborted, their algorithm does not make any attempt to find an optimal abort set as we have defined it.

Leung and Lai [5] discuss minimum cost recovery for multiprogramming, and not distributed, systems. They are concerned with aborting a set of jobs whose total cost is minimal among all sets that will resolve the deadlock. They give three heuristics, each of which in the worst case can abort \( O(n) \) times the optimal number of processes. Their algorithms abort processes in based on (1) the cost of the process; (2) the number of resources requested and resources held; (3) the number of resources held. All three algorithms run in time \( O(mn^2) \). Their simulations showed the second algorithm to perform the best, with ratios to the optimal ranging from about 1.2 to 1.4.

3 Deadlock Model

The WFG is input to our resolution algorithm. A reasonable, though not necessary, requirement is that the WFG contain no cycles of length two. A request by process \( P \) for resource \( R_1 \) that is held by process \( Q \) can check that \( Q \) is not waiting for some resource \( R_2 \) held by \( P \). This is possible if processes keep an updated list of those processes waiting for their resources. This avoids many common deadlocks.

We assume the deadlock detection algorithm begins when some process, called "the initiator, suspects that it is deadlocked [9]." This is called the timeout method [7] and is used in [7][13][14]. The reason for using the timeout method of detection is that the WFG is constructed only during deadlock detection, it is not maintained in part at each of the nodes.

The timeout method compares favorably to methods that store nonlocal portions of the WFG, or the entire WFG, at each process [15]. Those may require \( O(nm) \) storage at each process and \( O(m) \) messages per resource request to propagate the request. Likewise, the detection methods based on the timeout method can require \( O(nm) \) messages to detect a deadlock, but also may require \( O(m) \) messages per resource request if there is no deadlock [9][11]. It is open as to which method results in fewer messages in practice, but it can be seen that the timeout method sends fewer messages if the detection algorithm is invoked no more than once out of every \( n \) resource requests, on average.

As in Bracha and Toueg [13], we realize the possibility of up to \( k \) initiators. However, each detection instance in their algorithm runs independently, creating possible resolution problems shown in Figure 2. The possibility of two or more processes independently detecting the same deadlock creating inefficient resolutions was also cited in [1].

If \( I_1 \) and \( I_2 \) are both initiators, two inefficient resolutions may occur. \( I_1 \) can determine that it is in cycle \( C_1 \) and choose to abort some node \( P_1 \) in \( C_1 \). At
the same time I2 determines that it is in cycles C1 and C2 and chooses to abort node Pj in C1 and Pk in C2. Thus a process in C1 is unnecessarily aborted. Another possibility is for I2 to recognize that I1 has initiated a probe and to allow I1 to be the lone resolver. In that case, I1 will have no knowledge of C2 and after I1 aborts some Pj in C1, deadlock C2 will persist.

4 Resolution Heuristics

Our first heuristic, the edge cycle algorithm, resolves deadlock by aborting the process whose incoming and outgoing edges participate in the most cycles. We say an outgoing edge \((i, j)\) from \(P_i\) to \(P_j\) participates in a cycle if there is a path from \(P_j\) to \(P_i\). Likewise an incoming edge \((u, i)\) to \(P_i\) is in a cycle if there is a path from \(P_i\) to \(P_j\). If the abort resolves the deadlock, we are done. If not, we recompute the edge cycle sums and repeat the process. The algorithm is given in Figure 3, with \(G\) represented as an adjacency matrix.

**Input**: Wait-for-Graph \(G=(V, E), |V|=n, |E|=m\)

**Output**: Feedback Vertex Set

**Initialization**: Abort Set is empty; cycle:=false;

**repeat**

1. for each vertex \(v \in V\) do
   cycles[\(v\)]:=0;
2. \(G' := \text{Transitive Closure}(G)\);
3. for each vertex \(v \in V\) do begin
   3a. for each edge \((v,u) \in E\) do
       if \(G'[u,v]=1\) then
           cycles[\(v\)]:=cycles[\(v\)] + 1;
           cycle:=true;
   3b. for each edge \((u,v) \in E\) do
       if \(G'[v,u]=1\) then
           cycles[\(v\)]:=cycles[\(v\)] + 1;
           cycle:=true;
   end;
4. Abort Set := Abort Set \(\cup \{P \text{ such that }\)
   cycle_counts[P] is maximum\);
5. remove \(P\) from \(G\);
   until not cycle;

**Figure 3. Edge Cycle Resolution Algorithm**

The repeat loop is executed at most \(n-1\) times. Each iteration takes time \(O(n^3)\); the time to compute the transitive closure of \(G\). Thus the complexity of the algorithm is \(O(n^4)\). Alternatively, by considering edges, the algorithm can be implemented in \(O(m^2)\) time.

The second resolution algorithm to be considered is the max-cycle method. This algorithm removes the processes that are in the most cycles in the WFG. However, doing this requires that the cycles in the WFG be enumerated. This has been shown to be \#P-complete [16]. However, if the WFG has no more than \(O(n^x)\) cycles, for a constant \(x\geq 0\), the cycles can be enumerated in \(O(n^{2x+2})\) time [16]. Such graphs are termed polynomial cycle graphs (PCG), and finding an abort set in a PCG is the poly-abort set problem (PAS).

In our simulations, we place an upper bound of \(O(n^3)\) cycles in a PCG. We believe that, given an effective algorithm that detects deadlocks expeditiously - before the number of cycles becomes too large, these graphs are a reasonable model of deadlocks in many asynchronous computer systems.

In many (asynchronous) computer systems, one may expect that the number of resources a process is waiting on is not directly related to the number of processes in the system. In other words, beyond a certain number of processes in the system, the maximum number of resources a process can wait for is constant. Thus it is reasonable to assume in many cases, that WFGs have a (small degree) polynomial number of cycles.

We now give an algorithm to find an abort set in a PCG. The algorithm transforms the PAS problem to an instance of the Minimum Cover problem and uses the greedy algorithm of [17] to find a solution.

**Algorithm Polynomial Cycle Abort Set**

**Input**: WFG \(G=(V, E) \mid |V|=n\), \(G\) has \(m=O(n^x)\) cycles. Initially, \(A:=\{\}\)

**Output**: Abort set for \(G\)

\{ Transform PAS to Minimum Set Cover \}
1. **Enumerate** all the cycles in G, let these be \{c_1, c_2, ..., c_n\}
2. Let S = \{1, 2, 3, ..., m\}
3. Let F = \{s_p, s_{2p}, ..., s_{np}\}
4. Let \( s_i = \{j \mid v_i \in c_j\} \), 1 \leq i \leq m
   
   (Steps 5 through 10 solve Minimum Set Cover)
5. **if** S = \{\} **then** return((\(v_i \mid s_i \in A\), \(l(s_i)\nt\))
6. Let g = \( s_i \) such that \(|s_i|\) is maximum for 1 \leq i \leq m
7. Let \( A = A \cup g \)
8. Let \( S = S - g \)
9. **for** 1 \leq j \leq m
   
   10. go to 5

**Figure 4. Max-Cycle Resolution Algorithm for PCGs**

The instance of Minimum Set cover has the cycles of the PCG as sets, and as subsets has the set of cycles that each vertex is in. The greedy algorithm proceeds by choosing the vertex that covers the most cycles that are as of yet uncovered, stopping when all cycles have been covered by a vertex in the cover (abort) set.

Let A be the abort set obtained by the above algorithm. Let Min be an optimal abort set. Let \( k \) be the largest number of cycles that any vertex is in. Note that if \( k \leq 2 \), the minimum abort set can be found in polynomial time. From Johnson [17], we can show the following bounds about the ratio of the size of the abort set obtained by the approximation algorithm compared to the size of the minimum abort set.

\[
\frac{|A|}{|Min|} \leq \sum_{j=1}^{k} \frac{1}{j}
\]

More generally, without reference to the number of cycles that any vertex is in, we have the following:

\[
\frac{|A|}{|Min|} \leq \ln(n) + 1
\]

The running time of steps 1 through 4 is dominated by the time to enumerate the cycles in G, which is \( O(n^{2k^2}) \). The running time of steps 5 through 10 is \( O(mn \log n) \) [17]. Thus the total time to find the abort set for G with \( O(n^2) \) cycles is \( O(n^{2k^2}) \).

### 5 Simulation Model

Deadlock resolution heuristics were tested by simulating their effectiveness on random WFGs. The graphs were generated by providing as parameters the number of vertices in the graph and the probability that an edge occurs between any two vertices. Once generated, all cycles of length two were removed by eliminating (randomly) one of the two edges.

The resolution algorithms tested were:

1. the optimal -- found by exhaustive search.
2. the max-cycle algorithm.
3. the edge cycle algorithm.
4. based on number of resources held (similar to Leung and Lai's third heuristic [5]), it is called HOLD;
5. based on the number of resource requests, this is called the REQUEST heuristic as we abort processes based on the number of resources they are requesting;
6. based on the sum of the resources held and requested (similar to Leung and Lai's second heuristic [5]). This is called the SUM algorithm.

The graphs used in the simulations had ten, twelve, fifteen, and twenty vertices. Directed edges were created between pairs of vertices with probabilities of ten, fifteen, twenty, twenty-five, or thirty-five percent.

One hundred runs for twenty and fifteen vertex graphs and one thousand runs for the smaller graphs, were made with each set of parameters and resolution method. One hundred runs were used for the larger graphs as the exponential time used to find the optimal solution made running longer simulations impractical. Thus we generated a large sample of both relatively sparse and relatively dense graphs of varying sizes.

It is believed that these random graphs are an accurate model of system deadlocks. In a distributed system, each process is able to request any of the system resources -- each process is equally likely to be dependent on each of the other processes. Thus the WFG representing the resource state has no inherent structural characteristics and hence a random graph is viable model for a WFG.

### 6 Simulation Results

The results of the simulations are summarized in Figure 5 and Figure 6. The values in the tables are the
averages over the series of runs. The edge probability (Edge Prob) cited in the figures was the probability that an edge exists in the WFG, before cycles of length two were eliminated. The actual number of edges in the graphs (and that number as a percentage of the possible number of edges) is also given.

From the figures, it can be seen that the max cycle and the edge cycle deadlock resolution heuristics were more effective than the other general heuristics tested (the max-cycle method applies only to PCGs). On average the edge cycle method aborted from 1.23 to 1.30 times the optimal number of processes and the max-cycle method was, on average, no more than ten percent worse than the optimal solution.

In addition, between thirty-nine and fifty percent of all the deadlocks generated (on the more dense/"difficult" graphs) were resolved optimally by the edge cycle algorithm in the small graphs, and nineteen to thirty percent of the runs were optimal in the larger graphs. For all the graph sizes tested, between sixty and eighty percent of all resolutions aborted less than or equal to 1.4 times the optimal number of processes, and seventy-five to ninety percent of all resolutions aborted less than or equal to 1.6 times the optimal number of processes. The trend with the larger graphs seemed to be that fewer optimal solutions were found than in the small graphs, but more solutions were found with ratios to the optimal in the 1.1 to 1.2 range. Only ten to twelve percent of the resolutions aborting two or more time the optimal number of processes, and only about one percent of all the deadlocks generated caused the heuristic to abort more than twice the optimal number of processes. Leung and Lai's second algorithm performed better than their third algorithm, which corroborates their findings in [5]. However, their second algorithm on average aborted 1.55 times the optimal number of processes, about sixteen percent worse than our edge cycle heuristic, on average.

From the tables, it can be seen that the cycle enumeration algorithm (max-cycle) was more effective than any of the other heuristics tested. On average, the max-cycle method aborted from 1.00 to 1.09 times the optimal number of processes, which ranges from two to twenty-five percent better than the next best method.

Note that in larger graphs, the number of cycles will become super-polynomial if the graph is dense. In fact, for fifteen vertices, the graph with an edge probability of twenty-five percent had more than $n^3$ cycles. For twenty vertices, the number of cycles exceeded that threshold when the edge probability was twenty percent. Hence, the max-cycle table entries for graphs with thirty-five percent of possible edges were omitted, as enumerating the cycles on such dense graphs is not practical. For example, fifteen vertex graphs with thirty-five percent of possible edges averaged 15,383 cycles with many graphs having more than 22,000 cycles.

7 Conclusions

Optimal deadlock resolution is NP-complete, but ad hoc resolution methods can, in many cases, terminate a large number of processes unnecessarily. The cost of such process terminations and subsequent restart is high in terms of both time and turnaround-time for the user. Two resolution algorithms were presented that find optimal and near-optimal resolutions in a large percentage of deadlocks. The more general edge cycle heuristic chooses victims based on the number of incident edges that are involved in cycles in the wait-for-graph. It was shown in simulations on random wait-for-graphs to compare favorably to other resolution heuristics. The max-cycle algorithm finds abort sets in WFGs that have a polynomial number of cycles. It does so by enumerating the cycles in a WFG and removing the vertices that are in the most cycles. The max-cycle algorithm was shown in simulations to find abort sets that are on average no more than ten percent larger than the optimal abort sets.

REFERENCES


Figure 5. Simulation Results of Resolution Methods

Figure 6. Cumulative Percentages of Runs with Edge Cycle/Optimal Resolution Ratios ≤ Values Indicated