

Deadlock Resolution Techniques: An Overview

Pooja Chahar, Surjeet Dalal

Manav Rachna International University, Faridabad, Haryana
Manav Rachna International University, Faridabad, Haryana

Abstract- A deadlock occurs when there is a set of processes waiting for resource held by other processes in the same set. The processes in deadlock wait indefinitely for the resources and never terminate their executions and the resources they hold are not available to any other process. The occurrence of deadlocks should be controlled effectively by their detection and resolution, but may sometimes lead to a serious system failure. After implying a detection algorithm the deadlock is resolved by a deadlock resolution algorithm whose primary step is to either select the victim then to abort the victim. This step resolves deadlock easily. This paper describes deadlock detection using wait for graph and some deadlock resolution algorithms which resolves the deadlock by selecting victims using different criteria.

Index Terms-Deadlock, WFG, Processes, Resources, Release set, Transactions.

I. INTRODUCTION

A set of process is in a deadlock state if each process in the set is waiting for an event that can be caused by only another process in the set [1][2][3][15]. In other words, each member of the set of deadlock processes is waiting for a resource that can be released only by a deadlock process. None of the processes can run, none of them can release any resources, and none of them can be awakened.

A deadlock occurs when there is a set of processes waiting for resource held by other processes in the same set. The processes in deadlock wait indefinitely for the resources and never terminate their executions and the resources they hold are not available to any other process [3]. A deadlock lowers the system utilization and hinders the progress of processes. Also the presence of deadlocks affects the throughput of the system. The dependency relationship among processes with respect to resources in a distributed system is often represented by a directed graph, known as the Wait for Graph (WFG). In the WFG each node represents a process and an arc is originated from a process waiting for a resource to a process holding the resource.

In a distributed system, a deadlock occurs when there is a set of processes and each process in the set waits indefinitely for the resources from each other [15]. Therefore it is quite essential that a fast deadlock detection and resolution mechanism is applied otherwise the processes involved in the deadlock will wait indefinitely and will lower the system utilization and hinders the progress of processes.

A deadlock needs to be resolved timely because if not resolved, the deadlock size will increase with the deadlock persistence time as more processes will be trapped in the deadlock where a deadlock size is defined as the total number of blocked processes

(BP) involved in deadlock, where BP is the process that waits indefinitely on other processes. Because of deadlock none of the any processes involved can make any progress without obtaining the resources for which they are waiting.

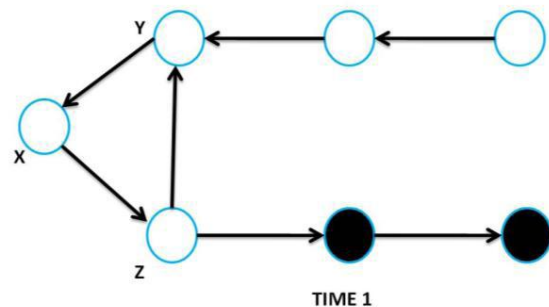


Figure 1: A few processes in deadlock

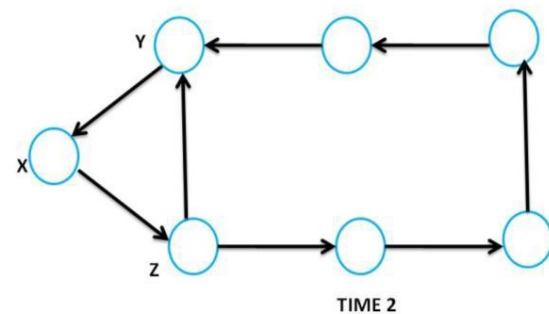


Figure 2: Increasing deadlock size as more processes trapped in deadlock

Because distributed systems are vulnerable to deadlocks, the problems of deadlock detection and resolution have long been considered important problem in such systems. Several models have been proposed for the processes operating in distributed system. As per the AND model, a process sits idle until all of the requested resources are acquired. In the OR model, a process resumes execution if any of the requested resources is granted. In the P-out-of-Q model also known as the generalized model, a process makes Q resource requests and remains blocked until it obtains any P resources. A generalized model is found in many domains such as resource allocation in distributed operating systems and communicating processes.

A deadlock is defined differently depending on the underlying model. Since a process becomes blocked if any of its resource requests is not granted, a deadlock in the AND model corresponds to a cycle in the WFG. In the OR model, the presence of a knot in the graph implies a deadlock. In the

generalized model a deadlock involves a more complex topology in the WFG. A cycle is a necessary but not sufficient condition for deadlock in this model.

II. WAIT FOR GRAPH

Deadlock detection is the process of actually determining that a deadlock exists and identifying the processes and resources involved in the deadlock. The basic idea is to check allocation against resource availability for all possible allocation sequences to determine if the system is in deadlocked state. Of course, the deadlock detection algorithm is only half of this strategy. Once a deadlock is detected, there needs to be a way to recover several alternatives exists:

- Temporarily prevent resources from deadlocked processes.
- Back off a process to some check point allowing preemption of a needed resource and restarting the process at the checkpoint later.
- Successively kill processes until the system is deadlock free. These methods are expensive in the sense that each iteration calls the detection algorithm until the system proves to be deadlock free. The complexity of algorithm is $O(N^2)$ where N is the number of proceeds. Another potential problem is starvation; same process killed repeatedly.

The simplest and easiest way to detect deadlock is wait for graph. A wait-for graph in computer science is a directed graph used for deadlock detection in operating systems and relational database systems. In computer science, a system that allows concurrent operation of multiple processes and locking of resources and which does not provide mechanisms to avoid or prevent deadlock must support a mechanism to detect deadlocks and an algorithm for recovering from them. One such deadlock detection algorithm makes use of a wait-for graph to track which other processes a process is currently blocking on. A wait for graph is a graph that consists of set of edges (E) and vertices (V). Processes are represented by vertices. In a wait-for graph, an edge from process P_i to P_j implies P_j is holding a resource that P_i needs and thus P_i is waiting for P_j to release its lock on that resource. A deadlock exists if the graph contains any cycles. The wait for graph scheme is applicable to a resource allocation system with multiple instances of each resource type.

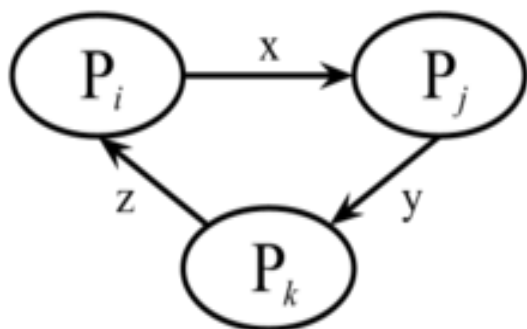


Figure 3: Wait for graph for deadlock detection.

In figure 3, P_i , P_j and P_k represents the processes in deadlock. An edge from P_i to P_j represents that P_i is waiting for resource x that is currently hold by P_j and so on.

III. DEADLOCK RESOLUTION

Deadlock detection is the process of actually determining that a deadlock exists and identifying the processes and resources involved in the deadlock. The basic idea is to check allocation against resource availability for all possible allocation sequences to determine if the system is in deadlocked state. The deadlock detection and resolution algorithm always require that transactions should be aborted .For this reason several issues must be carefully considered.

- 1) Aborts are more expensive than waits.
- 2) Unnecessary aborts result in wasted system resources.
- 3) Optimal concurrency requires that the number of aborted transactions be minimized

These factors must be considered so that the transaction being aborted will have the least impact on system performance and throughput[5]. Basically the deadlocks present in a system are detected by a periodic initiation of an effective deadlock detection algorithm and then resolved by a deadlock resolution algorithm and it is always tried that the resolution algorithm used does not cause any unnecessary aborts / roll backs.

The appropriate scheme for handling deadlocks in distributed systems is detection and resolution. A typical method to resolve deadlock is to select a proper victim. The victim is to abort itself for deadlock resolution.

The primary issue of deadlock resolution is to selectively abort a subset of processes involved in the deadlock so as to minimize the overall abortion cost. This is often referred to as the minimal abort set problem. The victim (aborted) processes need to cancel all pending requests and releases all acquired resources so that false deadlocks detection and resolution could be avoided.

Usually, the deadlocks are resolved by aborting deadlocked processes. Therefore, two facts have to be considered when analyzing the cost associated to deadlock resolution algorithms: the cost of detecting a deadlock and the time that the aborted processes have wasted. Deadlock situations when detected should be resolved as soon as possible but ensuring a minimum number of abortions and only those processes should be aborted which has been selected as victim. Thus, algorithms (safe-resolution algorithms) verifying the safety correctness criterion of resolving only true deadlocks should be designed

In fact the deadlock detection using wait for graph is safe detection algorithm and it is considered correct because they detect in finite time, all deadlock of the system and do not detect false deadlock[5]. Generally this algorithm doesn't take into account how a detected deadlock is resolved. It is only assumed that it is properly resolved. The algorithms do not explicitly model the resolution of detected deadlocks. Neither the system nor the code of the algorithm includes the effect of resolutions. Most of deadlock resolution algorithms abort or terminate the victim process. The only ways in which they differ is how they

select the victim. Most of the strategies of victim selection have been reviewed in the literature, the only drawback of such strategies is that it leads to abort of the victim, or they restart the victim which leads to wastage of resources, wastage of the work done by the aborted process, low throughput of system and it makes execution time of processes unpredictable. May be sometimes the aborted process have to be restarted in order to complete their work. Restarting a transaction is more expensive than waiting; therefore aborting a transaction needs to be avoided. In this paper algorithms for deadlock resolution have been discussed that uses different approaches for selecting a victim.

IV. RESOLUTIONALGORITHMS

Here we will discuss two algorithms for deadlock resolution which uses different criterion for victim selection

A. Resolution by using Timestamp

One of the most commonly used technique for deadlock resolution is timestamp based approach for selecting the victim. In this approach, a timestamp is allocated to each process as soon as it enters the system. The timestamp of the younger process is greater than the timestamp of older process. According to this approach, the victim is selected on this timestamps, the process with the higher timestamp is aborted, that is the youngest process is selected as the victim and is aborted in order to break the deadlock cycle. The goal behind choosing the youngest process as victim is that the youngest process would have used less resources and less CPU time as compared to older process. One problem with this technique is that it can cause starvation problem because every time a younger process is aborted which can starve the younger process from completion.

B. Resolution by using Burst time

Another approach for selecting a victim to break deadlock cycle is considering the burst time of each process. Burst time means the CPU time needed by any process for its execution. This can also be considered as one parameter for selecting a victim. The process with maximum burst time can be aborted in order to break cycle. The problem with this technique is that it can abort the process with high burst time which has been in the system for very long i.e. an older process with high burst time can be aborted which is inefficient approach.

C. Resolution by Degree

In a wait-for-graph for any system, the degree of any vertex denoting a process determines how many resources a process is holding and how many resources a process is requesting. There are two types of degrees in a directed WFG:

1. **In-degree:** In-degree means the number of edges coming to any node of WFG and it denotes number of request for resources held by a process.
2. **Out-degree:** Out-degree means the number of edges going out of a node in WFG denoting number of request for resources done by the node.

In resolution by degree, degree of each process is calculated and process having highest degree is aborted. Degree of any process can be calculated by taking sum of in-degree and out-degree.

D. Resolution by combination of Timestamp and Burst time

Another approach for selecting victim for deadlock is using both timestamp and burst time in combination. Select a process as victim which is younger and has high burst time for resolving deadlock. The advantage with this approach is younger process which will take maximum execution time will be aborted to allow processes with less execution time to complete first.

E. Resolution by combination of Burst time and Degree

Another combination for resolving deadlock is considering Burst time and Degree both for selecting a victim. Process with high burst time and high degree should be aborted that means a process which is having more resource request and will take high time to complete will be aborted. Although, there is still the problem of older process to be aborted but the advantage with this approach is aborting process with high burst time and high degree will release maximum resources needed for completion of other process with less execution time needed.

F. Resolution by combination of Degree and Timestamp

Taking degree and timestamp both in combination for resolving deadlock can prove to be another technique for deadlock resolution. A younger transaction with high degree will be aborted. The problem of starvation in considering only timestamp will be avoided in this case as degree of the node is also considered along with timestamp in order to select victim for resolving deadlock in the system.

G. Time Efficient Deadlock Resolution Algorithm

Deadlock is a major concern in a distributed system, since resources are shared among processes at sites distributed across a network. One of the most accepted methods of deadlock handling is detection and resolution. Both deadlock prevention and avoidance strategies are conservative solutions, whereas deadlock detection is optimistic[15]. In deadlock detection and resolution, deadlocks are allowed to occur[3][15]. Periodically, or on certain conditions, a detection algorithm is executed; if any deadlock state is found, resolution is undertaken. To resolve a detected deadlock, the system must abort one or more processes involved in the deadlock and release the resources allocated to the aborted processes. Here deadlock resolution with reusable resources is considered. In resolving a deadlock state, it is desirable to minimize the number of processes to abort to make the system deadlock-free. Concept of release set is introduced here. A release set is a set of one or more processes that can be

transaction, they form a deadlock cycle and none of them is being able to proceed ahead.

In the proposed deadlock resolution algorithm transaction, coordinator observes the scenario and it suspends T_{i+1} for some random t seconds and it releases resource R_{i+1} which is acquired by the requesting transaction T_i . It has been allotted the resource for the t seconds which is the time for which T_{i+1} has been suspended. T_i is supposed to utilize R_{i+1} and execute successfully in t seconds.

If T_i successfully executes before t seconds it sends a message to coordinator that it has successfully executed and to resume transaction T_{i+1} and gives its resource R_{i+1} back to T_{i+1} . If T_i is not able to complete its execution within t second coordinator preempts resource R_{i+1} from T_i and provides it back to T_{i+1} . The value R_{i+1} is the value partially updated by T_i . Now T_{i+1} will check whether T_i is still requesting for R_{i+1} . If it is requesting, T_{i+1} informs coordinator and is suspended again for some random t seconds and resource R_{i+1} is again allotted to T_i , T_i acquires it and resumes its execution and when completed before t seconds T_i informs coordinator to resume T_{i+1} and gives back resource R_{i+1} to T_{i+1} .

Similarly coordinator blocks T_n for some random t seconds and it releases resource R_n which is acquired by the requesting transaction T_{n-1} . It has been allotted the resource for the t seconds which is the time for which T_n has been suspended. T_{n-1} is supposed to utilize R_n and execute successfully in t seconds.

If T_{n-1} successfully executes before t seconds it sends a message to coordinator that it has successfully executed and to resume transaction T_n and gives its resource R_n back to T_n . If T_{n-1} is not able to complete its execution within t seconds coordinator preempts resource R_n from T_{n-1} and provides it back to T_n . The value of R_n is the value partially updated by T_{n-1} . Now T_n checks whether T_{n-1} is still requesting for R_n . If it is requesting T_n informs coordinator and is suspended again for some random t seconds and resource R_n is again allotted to T_{n-1} , T_{n-1} acquires it and resumes its execution and when completed before t seconds T_{n-1} informs coordinator to resume T_n and gives back resource R_n to T_n .

V. CONCLUSION

Deadlock is a major problem in operating systems. However there are several techniques to deal with deadlock such as deadlock avoidance, prevention etc. but still deadlock can occur. The only way to deal with deadlock when it occurs is to detect and resolve it as soon as possible. Several techniques to resolve deadlock are mentioned above. One can use any of the above technique to resolve deadlock and deadlock will be resolved.

REFERENCES

- [1]. Yan Cai, k.Zhai, Shngru wu, W.k. Chan, "Synchronizing Threads Globally to Detect Real Deadlocks For Multithreaded Programs", ACM 2013
- [2]. Jeremy D.Buhler, Kunal Agrawal, Peng Li, Roger D. Chamberlain, "Efficient Deadlock Avoidance For Streaming Computation With Filtering", 2012 ACM, February 2012.
- [3]. Kunwar Singh Vaisla, Menka Goswami, Ajit Singh, "VGS Algorithm - an Efficient Deadlock Resolution Method",

International Journal of Computer Applications (0975 – 8887), Vol.44- No. 1, April 2012.

- [4]. Selvaraj Srinivasan, R. Rajaram, "A decenralized deadlock detection and resolution algorithm for generalized model in distributed systems", january 2011.
- [5]. Selvaraj Srinivasan, R. Rajaram, "A decenralized deadlock detection and resolution algorithm for generalized model in distributed systems", January 2011.
- [6]. Iryna Felko, TU Dortmund, "Simulation based Deadlock Avoidance and Optimization in Bidirectional AGVS", 2011 ACM, march 2011.
- [7]. Prodromos Gerakios, Nikolaso Papaspyrou, Konstantinos, Vekris, "Dynamic Deadlock Avoidance in System Code Using Statically Inferred Effects", 2011 ACM, October 2011.
- [8]. Peng Li, kunal agrawal, JeremyBuhler, Roger D. Chamberlain, "Deadlock Avoidance for Streaming Computations with Filtering", 2010 ACM, June 2010.
- [9]. Srinivasan Selvaraj, R. Ramasamy, "An Efficient Detection and Resolution of Generalised Deadlocks in Distributed Systems", International Journal of Computer Applications, vol 1- no. 19
- [10]. Hari K. Pyla, Srinidhi Varadarajan, "Avoiding Deadlock Avoidance", 2010 ACM, September 2010.
- [11]. Pallavi Joshi, Mayur Naik, "A randomized Dynamic Program Analysis Technique for Detecting Read Deadlocks", 2009 ACM, June 2009.
- [12]. Yibei Ling, Shigang Chen and Cho-Yu Jason Chiang, "On Optimal Deadlock Detection Scheduling", IEEE Transactions On Computers, Vol. 55, No. 9, September 2006.
- [13]. Lee, S., "Fast, Centralized Detection and Resolution of Distributed Deadlocks in the Generalized Model", IEEE Trans. On Software Engineering, Vol. 30, NO. 9, 561-573, 2004
- [14]. D. Manivannan and Mukesh Singhal, "An Efficient Distributed Algorithm for Detection of Knots and Cycles in a Distributed Graph", IEEE Transactions on Parallel And Distributed Systems, Vol. 14, No. 10, October 2003.
- [15]. Terekhov, T. Camp, "Time efficient deadlock resolution algorithms", June 1998.