Webology (ISSN: 1735-188X)
Volume 14, Number 2, 2017

Comparative Studies Of Strategies Used In Deadlock Detection And Resolution Using Semaphore Based Dining Philosophers Problem

N.K SINGH[1] , ANURAG SINHA[2]

[1]Department of computer science, BIT Mesra.

[2]Department of computer science and IT, UG SCHOLAR Amity University
Jharkhand, Ranchi, Jharkhand (India)

OBJECTIVES

The project objective is to ensure the deadlock resolution and to do comparitive study algorithms used in deadlock detection and resolution and thereby reaching out of a concrete technique to implement it and the best suited strategy to deal with deadlock problem 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. In fact most of the deadlock detection algorithms in literature are safe detection algorithms and they are considered correct because they detect infinite sorts.

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 victim

LITERATURE REVIEW

The occurrence of deadlocks should be controlled effectively by their detection and resolution, but may sometimes lead to a serious system failure. After implying an efficient 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 transaction or cause it to rollback. This step resolves deadlock but is not efficient one. This paper proposes a new deadlock resolution algorithm which doesn‟t cause any aborts /roll backs in fact it is based on the mutual cooperation of

transactions and a random number representing time duration for which the process holding the resource will be suspended.

CONCEPT OF THE DEADLOCK

A deadlock is a condition where two or more users are waiting for data, locked by each other. Oracle automatically detects a deadlock and resolves them.

-Deadlock occurs when transactions executing at the same time lock each other out of data that they need to complete their logical units of work.

-Deadlock is a situation where a group of processes are all blocked and none of them can become unblocked until one of the other becomes unblocked.

In a multiprogramming environment, several processes may compete for a finite number of resources. A process requests resources; if the resources are not available at that time, the process enters a waiting state. Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes. This situation is called a deadlock.

Consider a case where two different processes want to be allocated on the same resource (say printer) at a particular time. If both the processes requests for the same resource then the system will come under the state of deadlock because a single resource can attend only one process at a time. In other words, a printer can print only one process (document) at a time.

Some basic points on deadlock

Permanent blocking of a set of processes that either compete for system resources or communicate with each other

1. Involves conflicting needs for resources by two or more processes
2. No efficient general solution.

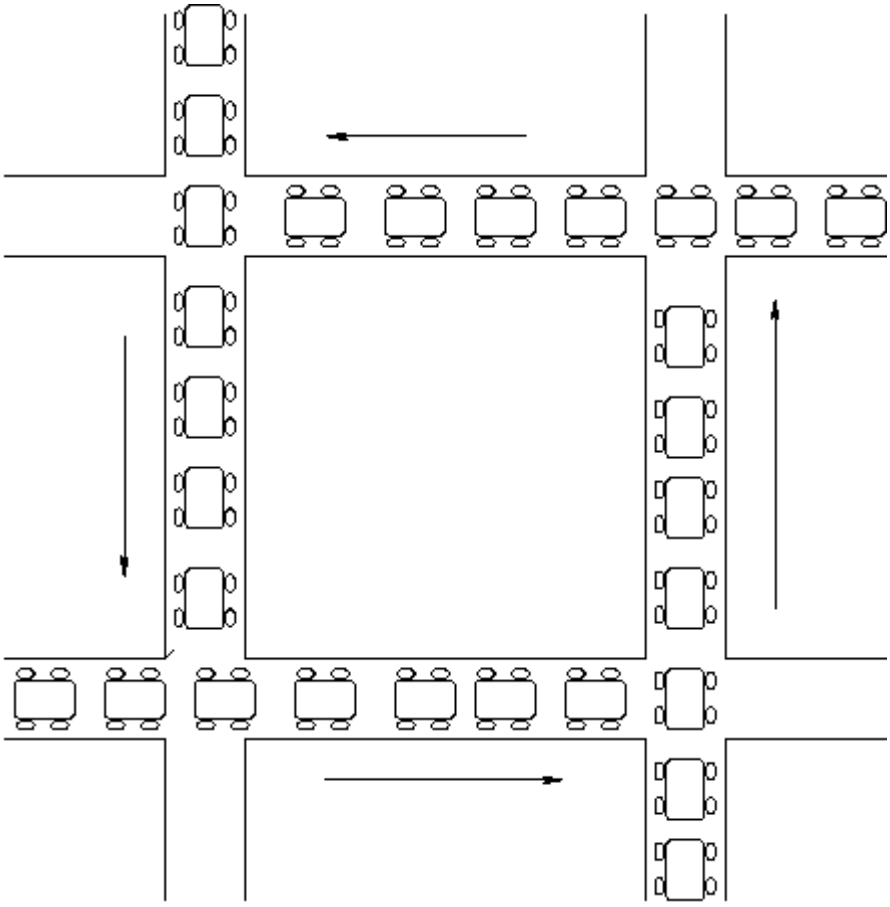
Necessary Conditions for Deadlock

The 4 Necessary Conditions for Deadlock

1. Exclusive access (mutual exclusion): only one process may use a resource at a time
2. Wait while holding (hold-and-wait):A process can continue to hold a resource while requesting another.
3. No preemption: A process cannot be forced to give up resources before it chooses to give them up.
4. Circular wait: There is a cycle of hold-and-wait relationships.

In order for there to be a deadlock, all of the above conditions must be true. You can observe that they are true in the examples we considered. Richard Holt, in a PhD dissertation published in the 1970's, showed that these conditions must apply in any deadlock. Informally, by looking at each condition and convincing yourself that if the condition is not true, there is no deadlock.

Example: Traffic gridlock is an everyday example of a deadlock situation.



DEADLOCK SYSTEM MODEL

A system consists of a finite number of resources to be distributed among a number of competing processes. The resources are partitioned into several types, each consisting of some number of identical instances. Memory space, CPU cycles, files, and I/O devices (such as printers and DVD drives) are examples of resource types. If a system has two CPUs, then the resource type CPU has two instances. Similarly, the resource type printer may have five instances.

A process must request a resource before using it and must release the resource after using it. A process may request as many resources as it require

es to carry out its designated task. Obviously, the number of resources requested may not excceed the total number of resources available in the system. In other words, a process cannot request three printers if the system has only two.

Under the normal mode of operation, a process may utilize a resource in only the following sequence:

1. **Request:** The process requests the resource. If the request cannot be granted immediately (for example, if the resource is being used by another process), then the requesting process must wait until it can acquire the resource.
2. **Use:** The process can operate on the resource (for example, if the resource is a printer, the process can print on the printer).
3. **Release:** The process releases the resource

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.

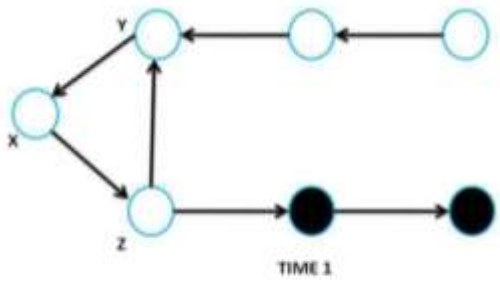


Figure1: A few processes in deadlock

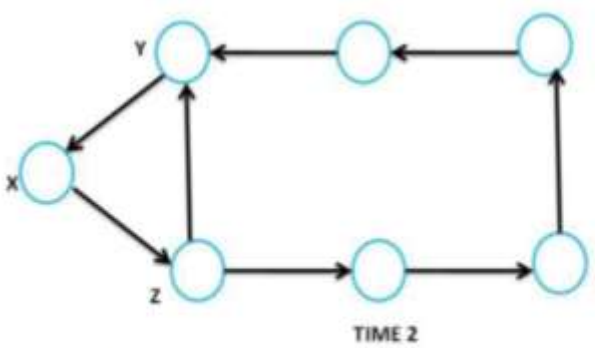


Figure 2: Increasing deadlock size as more processes trapped in deadlock

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.

RESEARCH METHOLOGIES

1.DEADLOCK DETECTION TECHNIQUE:- 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(N2) 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 Pi to Pj implies Pj is holding a resource that Pi needs and thus Pi is waiting for Pj 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

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

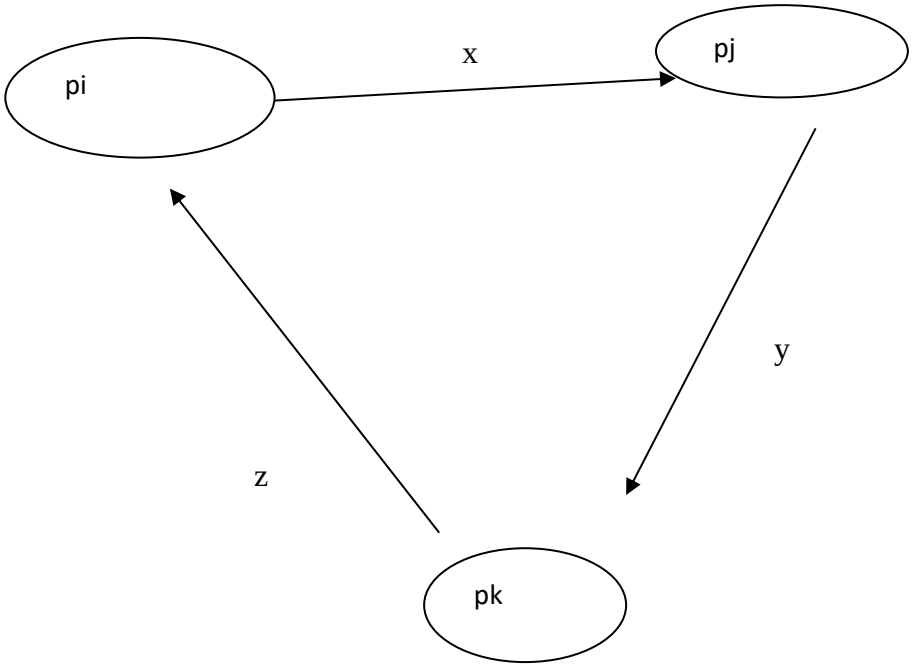


fig-3

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.

DEADLOCK RESOLUTION ALGORITHMS

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.

VGS ALGORITHM FOR DEADLOCK RESOLUTION

This section describes the solution to deadlocks in distributed systems i.e. VGS Algorithm an efficient deadlock resolution algorithm. In a distributed system if deadlock is detected at a site, then the site coordinator can apply VGS algorithm to resolve the deadlock[15]. This algorithm is based on the mutual cooperation of the transactions and is described as follows:

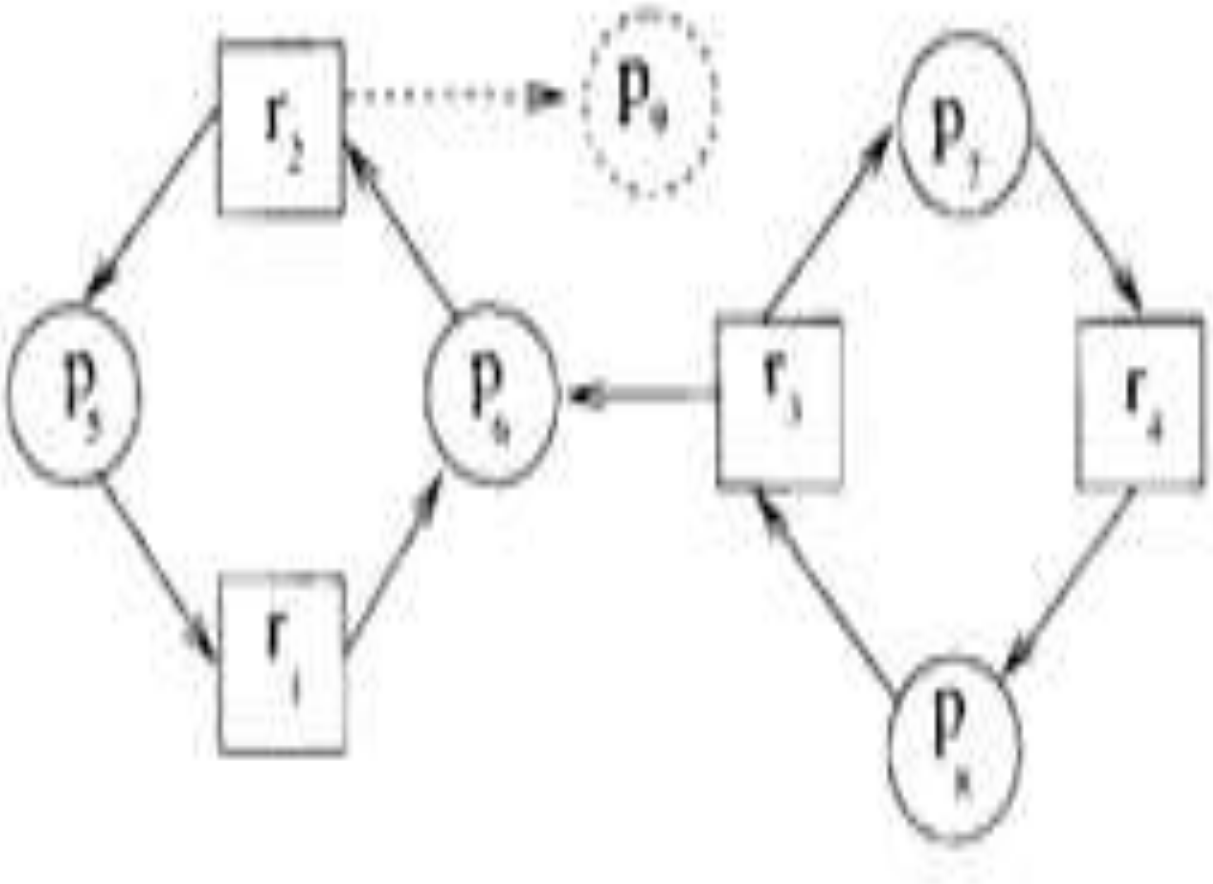
Ti REQUESTS Ri+1 Ti+1 REQUESTS Ri+2 .
. Tn-1 REQUESTS Rn TN REQUESTS Ri

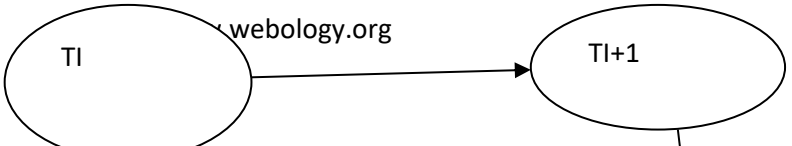
Suppose Ti, Ti+1, Ti+2……Tn are the transactions involved in a deadlock. They form a deadlock cycle such that Ti holds resource Ri, Ti+1 holds resource Ri+1, Ti+2 holds resource Ri+2……..Tn holds Rn and Ti is requesting for resource Ri+1 , Ti+1 is requesting for resource Ri+2 ……,Tn is requesting for Ri. Since each transaction is holding a resource and waiting indefinitely for other resource held by the other 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 Ti+1 for some random t seconds and it releases resource Ri+1 which is acquired by the requesting transaction Ti. It has been allotted the resource for the t seconds which is the time for which Ti+1 has been suspended. Ti is supposed to utilize Ri+1 and execute successfully in t seconds.

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

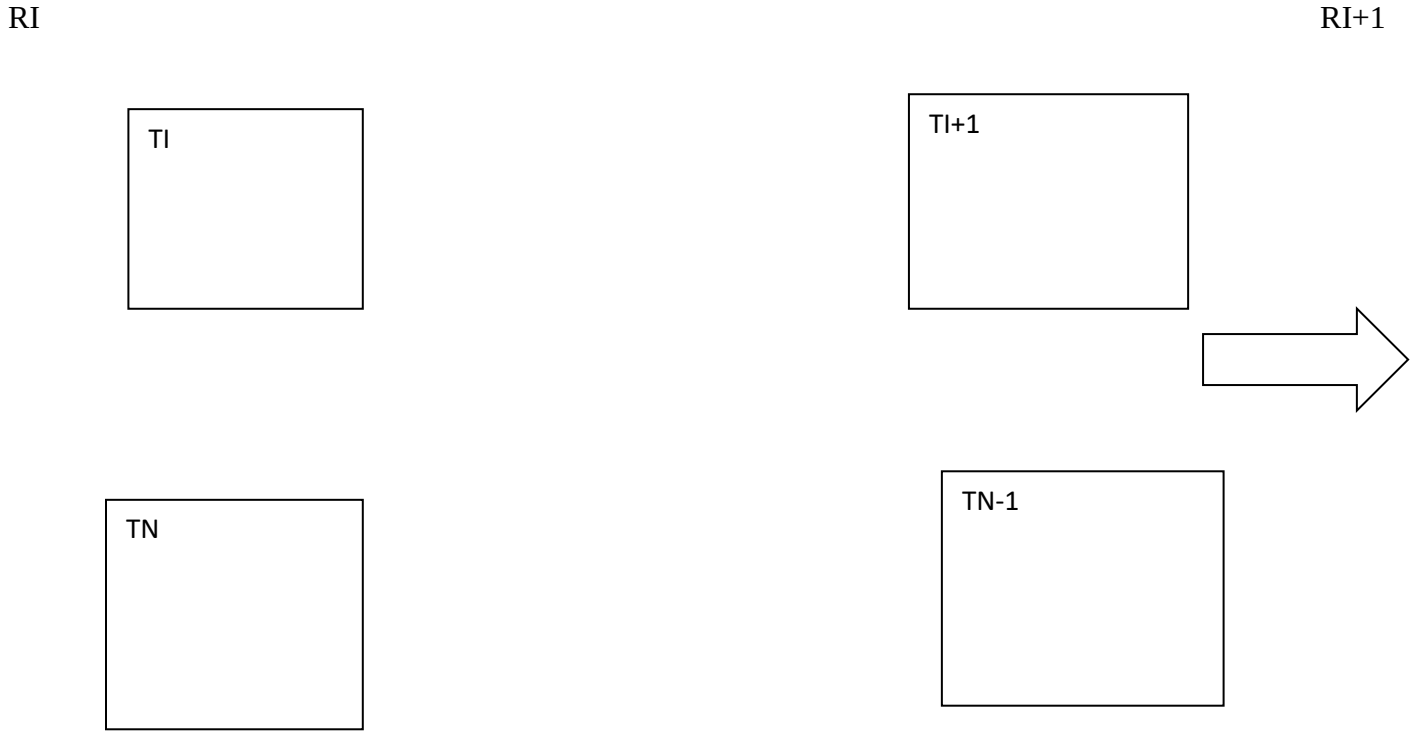
Similarly coordinator blocks Tn for some random t seconds and it releases resource Rn which is acquired by the requesting transaction Tn-1. It has been allotted the resource for the t seconds which is the time for which Tn has been suspended. Tn-1 is supposed to utilize Rn and execute successfully in t seconds. If Tn-1 successfully executes before t seconds it sends a message to coordinator that it has successfully executed and to resume transaction Tn and gives its resource Rn back to Tn. If Tn-1 is not able to complete its execution within t seconds coordinator preempts resource Rn from Tn-1 and provides it back to Tn. The value of Rn is the value partially updated by Tn-1. Now Tn checks whether Tn-1 is still requesting for Rn. If it is requesting Tn informs coordinator and is suspended again for some random t seconds and resource Rn is again allotted to Tn-1, Tn-1 acquires it and

resumes its execution and when completed before t seconds Tn-1 informs coordinator to resume Tn and gives back resource Rn to Tn.





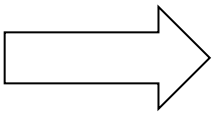
A deadlock cycle



TI+1 SUSPENDS FOR RANSOM T SEC..

Transaction Ti+1, Tn suspended and release resources

RN TN SUSPENDS FOR RANDOM T SEC
RN+1



RI,RI+1

RN,RN+1

TI,TN-1 EXECUTING SUCESSFULLY,NO DEADLOCK

FIG- **Ti, Tn-1 executing successfully.**

CONCLUSION

In this we presented deadlock resolution algorithm which resolves deadlocks effectively. As the paper describes in this algorithm the transactions resolve deadlock with the mutual cooperation of each other. Transaction Ti+1 and Tn suspend themselves and let other transactions proceed successfully and continuously co-operate them till they are not able to commit successfully. As compared to other resolution algorithms which cause abort or rollback it does not cause any such aborts or rollbacks, which proves its effectiveness. In the proposed algorithm the distributed system‟s site coordinator manages its own transactions and resolves any deadlock when detected.

Deadlock is a major problem in operating systems. However there are several techniques to dead 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. Smith, A. B., & Johnson, C. D. (2019). A Comparative Study of Air and Liquid Cooling Solutions for High-Performance CPUs. International Journal of Computer Cooling, 12(2), 45-62.

2. Brown, E. R., & Williams, J. M. (2020). Thermal Management Techniques for Modern Multi-Core Processors. Proceedings of the IEEE Symposium on High-Performance Computing, 178-185.

3. Lee, S. H., Kim, T. H., & Park, H. S. (2018). Evaluation of Phase-Change Materials for Advanced CPU Cooling. Journal of Thermal Science, 25(4), 315-325.

4. Chen, X., Li, Y., & Zhang, L. (2017). Performance Analysis of Heat Pipe-Assisted CPU Cooling Systems. International Conference on Electronics Cooling, 67-72.

5. Kumar, R., Singh, M., & Gupta, A. (2019). Comparative Study of Active and Passive Cooling Techniques for Overclocked CPUs. Journal of Computer Hardware Engineering, 8(2), 89-102.

6. Wang, Q., Li, Z., & Zhang, P. (2021). Analysis of Liquid Metal Thermal Interface Materials for Improved CPU Cooling Performance. Applied Thermal Engineering, 185, 116271.

7. Patel, S., & Smith, G. (2018). Experimental Investigation of the Impact of Fan Configurations on CPU Cooling. Proceedings of the ASME International Mechanical Engineering Congress and Exposition, 536-543.

8. Johnson, R. W., & Miller, A. F. (2020). Enhancing CPU Cooling Efficiency Using Microchannels and Nanofluids. Journal of Electronic Cooling and Thermal Control, 15(3), 120-134.