MATEC Web of Conferences **57**, 02013 (2016) DOI: 10.1051/matecconf/20165702013
ICAET-2016

Deadlock Prevention Algorithm in Grid Environment

Deepti Malhotra

Department of Computer Science & IT & Central University of Jammu

Abstract. Deadlock is a highly unfavourable situation, the deadlock problem becomes further complicated if the underlying system is distributed. Deadlocks in distributed systems are similar to deadlocks in single processor systems, only worse. They are harder to avoid, prevent or even detect. They are hard to cure when tracked down because all relevant information is scattered over many machines.In deadlock situations the whole system or a part of it remains indefinitely blocked and cannot terminate its task. Therefore it is highly important to develop efficient control scheme to optimize the system performance while preventing deadlock situations.In this research paper, a new deadlock prevention algorithm have been offered with the emergence of grid computing. The main objective of paper is to prevent deadlock problem in Grid environment in order to preserve the data consistency and increase the throughput by maximizing the availability of resources and to ensure that all the resources available in the grid are effectively utilized.

1 INTRODUCTION

Deadlocks are important resource management problem in distributed systems because it can reduce the throughput by minimizing the available resources. In distributed systems, a process may request resources in any order, which may not know a priori, and a process can request a resource while holding others. If the allocation sequence of process resources is not controlled in such environments, deadlock can occur. A deadlock can be defined as a condition where a set of processes request resources that are held by other processes in the set. Deadlock can be dealt with using any one of the following three strategies: *deadlock prevention, deadlock avoidance, and deadlock detection*. Deadlock prevention is commonly achieved by either having a process acquire all the needed resources simultaneously before it begins execution or by pre-empting a process that holds the needed resource. In the deadlock avoidance approach to distributed systems, a resource is granted to a process if the resulting global system is safe. Deadlock detection requires an examination of the status of the process-resources interaction for the presence of a deadlock condition. To resolve the deadlock, we have to abort a deadlocked process.Deadlock refers to the coordination and concurrency problem where two or more processes are waiting indefinitely for the release of a shared resource [1]. The deadlock problem involves a circular waiting where one or more transactions are waiting for resources to become available and those resources are held by some other transactions that are in turn blocked until resources held by the first transaction are released. [2, 3, 4].Deadlock processes never terminate their executions and the resources held by them are not available to any other process.

Resource Vs. Communication deadlock.
Two types of deadlock have been discussed in the literature: resource deadlock and communication deadlock. In resource deadlocks, processes make access to resources (for example, data objects in database systems, buffers in store-and forward communication networks). A process acquires a resource before accessing it and relinquishes it after using it. A process that requires resources for execution cannot proceed until it has acquired all those resources. *A set of processes is resource*- deadlocked if each process in the set requests a resource held by another process in the set. In communication deadlocks, *messages are the resources* for which processes wait. Reception of a message takes a process out of wait - that is, unblocks it. *A set of processes is communication*- deadlocked if each process in the set is waiting for a message from another process in the set and no process in the set ever sends a message. To present the state of the art of deadlock detection in distributed systems, this article describes a series of deadlock detection techniques based on centralized, hierarchical, and distributed control organizations.titles.

The rest of the paper is organized as follows. Section 2 presents the related work. Existent deadlock prevention algorithms are also discussed in this section. Section 3 provides the details of the proposed deadlock prevention algorithm Section4 also provides the results of the experiment carried out using our own grid simulated

[a] Corresponding author: deepti433@yahoo.com

environment.Finally Section5 concludes the paper by summarizing the contributions and future works.

2 RELATED DEADLOCK PREVENTION ALGORITHM IN GRID

Deadlock detection, resolution techniques and deadlock avoidance technology typically require pre empting resources, aborting processes or centralized resource management so they are inappropriate for many distributed real time systems that require the execution time of processes be predictable [5,6]. Therefore it would be more efficient if we look for an efficient deadlock prevention mechanism for distributed real time systems. The deadlock avoidance strategies make system performance to suffer [7], although deadlock detection may be effective, but it costs a lot in business transaction services [8] and detection of false deadlocks leads to wastage of resources. However deadlock resolution mechanisms are there but many real time applications are not well suited for run time deadlock resolution [9].

2.1 Replica Based local Deadlock Prevention

Replica based mechanism is also used to prevent local deadlocks. The basic idea is to produce a replication of the resource when more than one participants request it [8][10].Here the intelligent resource manager is used which is responsible for allocating resources for every applicant. Every transaction has a unique transaction id, including sub-transactions [8]. When a sub-transaction manager receives instruction from parent transaction manager/coordinator, it keeps the root transaction id passed by its parent and produces its own sub-id. So every participant knows the root id and we can distinguish if two sub-transactions belong to the same nested transaction. If two participants have the same root id, we don't care which level they are on.

In this type of mechanism, resource manager RM first receives $T_{1's}$ request and then T_2 for the same resource R. Step one, RM immediately caches the root transaction id from T_1, then it takes over $T_{M_{1's}}$ request and sets a lock on R. Soon T_2 comes. T_2 asks for RM to request R but it's already occupied by T_1. RM should not reject $T_{2's}$ request but checks its root transaction id first. It finds that more than one sub-transactions belong to the same local nested transaction, so it caches $T_{2's}$request and then creates a duplication of resource R (noted as R'). From now on, all requests from sub-transactions with the same root id should operate on resourceR'). But this time T_1will not hold its lock until phase two. In fact, RM will hold R's lock for root transaction and gives a new lock of R' toT_1. T_1 immediately commits its work on R' and returns prepared information to the coordinator if no exception occurs. Now RM takes $T_{2's}$ turn to occupy resource $R'$and work on it. After T_2 commit, the coordinator sends the second phase instruction to commit all of their works. T_1 or T_2 will pass this decision to RM. RM writes R' back to R and release its lock. At the end, all the participants return their final status to coordinator. On the other hand, if one of the participants cannot commit their work on replica R'), it should vote negative information to the coordinator. Then the coordinator notices all the participants to rollback their work. When RM receives this decision, it simply discards R').and release the lock on R.

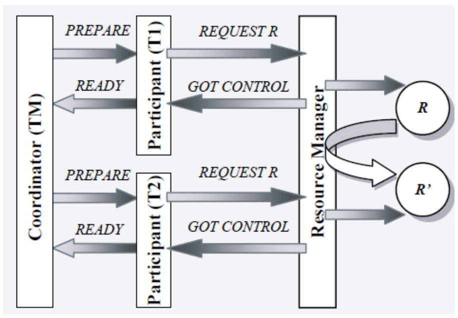


Figure 1. Replica based mechanism for guaranteed deadlock [8].

2.2 Timestamps Based Restart Policy for Global Deadlock prevention

In this section, we describe global deadlock in distributed transaction environment. This is a more frequently discussed topic, for it is a system level problem (e.g. competing hardware devices, computing resource or database operation). Standard technology of deadlock avoidance expects sequentially resource access. For some resources, system could declare the most amount of requirements in advance. This is a pessimistic static allocation algorithm that needs to exploit prior knowledge of transaction access patterns. If deadlock is allowed (e.g. it's rare to happen), detection and resolution are the main issues we should consider [11]. Timeout-based detection guesses that a deadlock has occurred whenever a transaction has been blocked for a long time. Wait-for graph is often used that if a circle exists in the graph, the transactions are regarded as deadlocked. But it's not easy to implement on distributed nodes. To resolve deadlock, we should choose a blocked transaction to be removed. For timeout-based detection, the blocked transaction can simply abort itself if a timeout period reaches. The best way of breaking all circles in a wait-for graph is to find a type of transactions with minimum conflicting cost. For example, we can choose the transaction that has done the least amount of work. In distributed transaction environment, business transactions are usually provided with different functional services. Each service is independent and already knows what resources it would request. That is to say, for every transaction, it is appropriate to exploit prior knowledge of related resources. Based on this premise, it has been prove that pre-assigning required resources should be a much efficient way to avoid global deadlocks.

To follow the two-phase commit protocol, we add a new phase called pre-check before every participant could prepare their work. At the first stage, the coordinator delivers all the user's requests to the participants, then these participants communicate with each resource managers to check the resources if they are available. If yes, the participants will hold them at the same time and return OK to the coordinator. It can be seen as a try-lock phase that means a transaction can go on if the resource lock is obtained, otherwise it should return false. After receiving all positive feedback from participants, the coordinator will enter in the standard two-phase commit process.

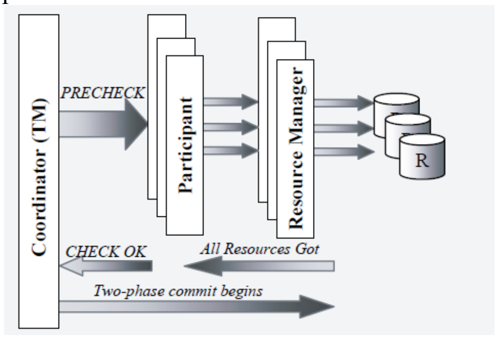


Figure 2. Global Deadlock Resolution [8].

2.3 VGS Algorithm For Deadlock Prevention

The proposed algorithm requires that the prior knowledge of the resources is necessary. Although most of the researchers do not appreciate the concept of prior knowledge of resources but in distributed applications where the transactions follow two phase commit protocol the prior knowledge of resources would be very beneficial. The prior knowledge of resources would let the transaction to be sub divided into sub transactions and distribute them at different sites as per the availability of resources as such the hold and wait condition would not occur. Most of the distributed transaction applications use two phase commit protocol to coordinate sub transactions [8]. The sub transactions need to lock the required resources before updating data to ensure that the transactions will commit at all sites or at none of them. The proposed algorithm prevents the deadlock in following ways:

1. The pre check phase, in which sub transactions are divided only when all of the required resources are available and free. As such none of any transaction will be waiting for resource. When they will be initiated at sites they will not be waiting for any resource. Hold and wait condition has been eliminated therefore no deadlock.

2. The transaction will be divided into sub transactions only when resources are available and will commit and the sub transactions (not of same transaction) requesting for same resource will execute in pipeline fashion. As such all of the sub transactions are active they are not in waiting state. Sites will be less populated by those transactions which are in waiting or requesting mode there by reducing the chances of formation of deadlock cycles. All of the transactions will be in some phase out of the four phases (pre check, coordinator, phase 1, and phase 2) and executing the algorithm effectively reduces the waiting time of the transactions requesting for the same resource.

3 PROPOSED DEADLOCK PREVENTION ALGORITHM

The Algorithm that we are proposing and trying to implement is based on the idea of No-preemption for deadlock prevention, existence of deadlocks in a smaller network could be overcome easily, but if we are dealing with a Grid environment, it could be technically and economically infeasible to deal with such a situation. That will in turn result into the blockage of Network packets in and around the network. In a distributed environment, resources as well as processes are geographical distributed. A process requests the resources; if the resources are not available at that time, the process enters into a wait state. Waiting processes may never again change state, because the resources they have requested are held by some other waiting processes. So our algorithm may provide a provision for efficient allocation of resources such that deadlocks never occur in a grid environment.

The simulation experiment provides us of a general idea how this could be realized for a Grid Environment. Our algorithm relates to allocation of resources by following the three protocols of no preemption condition for deadlock prevention and following those three protocols step by step allocation of resources as per the protocols. Number of processes and number of resources to be allocated as input by the user are arranged in the form of a matrix of processes and available resources. Similarly, resources to be allocated to a process or processes are again arranged in a matrix form. The algorithm restricts the occurrence of deadlock by considering the constraints of the program.

The proposed algorithm prevent deadlock in grid environment by considering no preemption condition. Grid is a collection of large database and it is prone to deadlock. In grid environment the resources as well as processes are geographically distributed. The deadlock prevention algorithm work as:

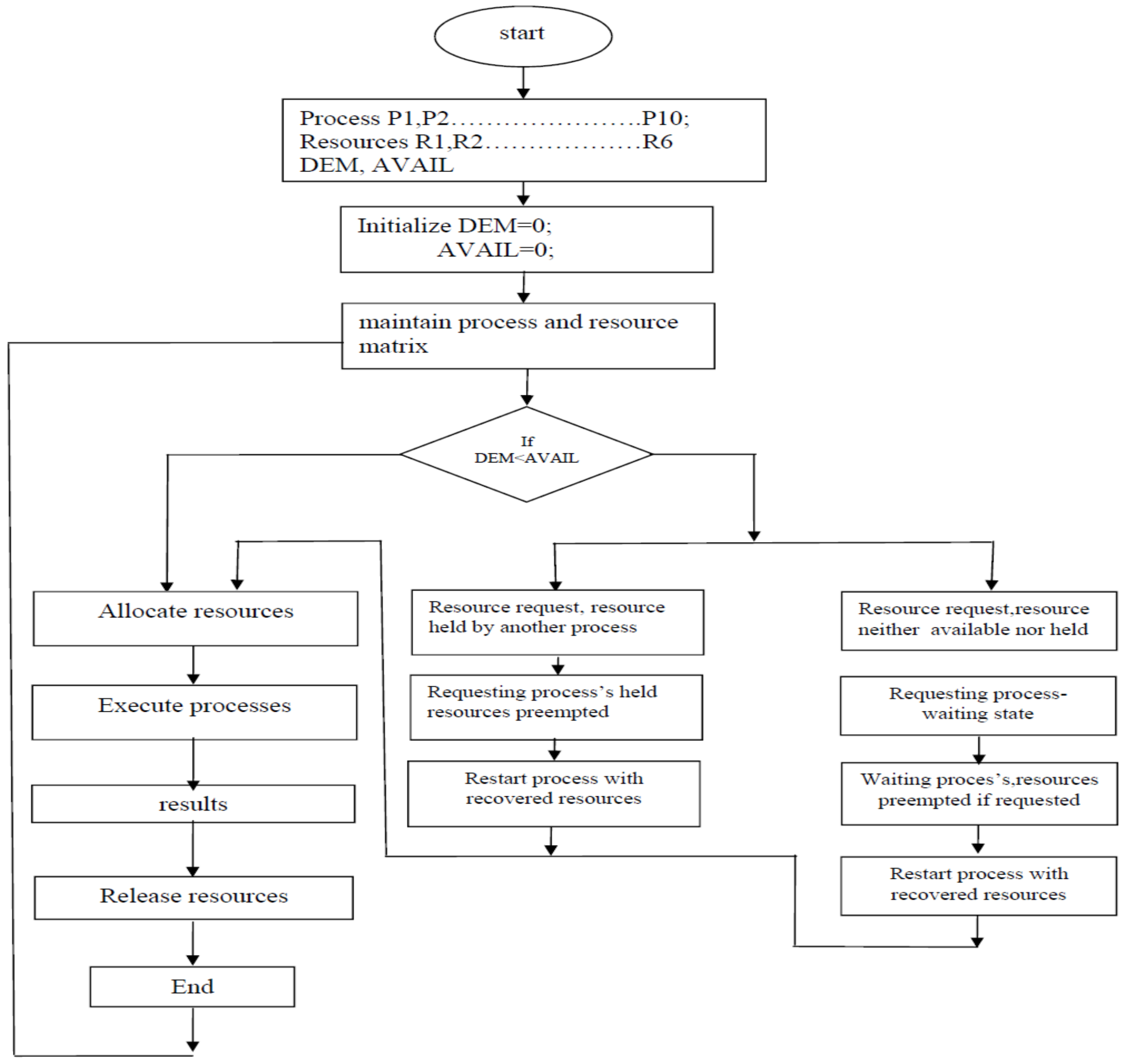
1. if demand is less than availability then the resources are allocated.

2. if demand is greater than availability then two conditions arises-

- if a process is holding some resources and request a resource which is held by some other process, then all the resources currently held are preempted.The pre-empted resources are added to the list of resources for which the process is waiting. The process will restart again only if it regains its old resources, as well as the new ones that it is requesting.
- If the resources are neither available nor held by a waiting process, the requesting process must wait. While it is waiting, some of its resources may be preempted, but

only if another process requests them. A process can be restarted only when it allocated the new resources it is requesting and recovers any resources that were preempted while it was waiting.
This algorithm ensures that resources are allocated to a process for a time span to avoid occurrence of deadlock. For example if the time span is 10s then it is available to the process for 10 seconds. After 10 sec it is available to another requesting process.

3.1 Flowchart


start
Process P1,P2.....................P10;
Resources R1,R2.................R6
DEM, AVAIL
Initialize DEM=0;
AVAIL=0;
maintain process and resource matrix
If DEM<AVAIL
Allocate resources
Execute processes
results
Release resources
End
Resource request, resource held by another process
Requesting process's held resources preempted
Restart process with recovered resources
Resource request,resource neither available nor held
Requesting process- waiting state
Waiting proces's,resources preempted if requested
Restart process with recovered resources

3.2 Assumptions of algorithm

1. Number of processes-10 for easy understanding

2. Number of resources-6

3. Consider only 3 conditions of no pre-emption protocol that is-

- If a process requests some resources, we first check whether they are available. If they are, we allocate them. If they are not, we check whether they are allocated to some other process that is waiting for additional resources. If so, we preempt the desired resources from the waiting process and allocate them to the requesting process.
- If a process is holding some resources and request a resource which is held by some other process, then all the resources currently held are pre-empted. The pre-empted resources are added to the list of resources for which the process is waiting; the process will restart again only if it regains its old resources, as well as the new ones that it is requesting.
- If the resources are neither available nor held by a waiting process, the requesting process must wait. While it is waiting, some of its resources may be preempted, but only if another process requests them. A process can be restarted only when it allocated the new resources it is requesting and recovers any resources that were preempted while it was waiting.

4. Input only integer values.

5. Message, unsafe state if there is chance of deadlock.

3.3 Pseudocode

As per flowchart

1. We assume number of processes not to exceed 10.

2. Also, the number of resources must not exceed 6, if either the number of processes or resources exceeds this limit, there will be an warning message.

3. Initialize the variables DEM and AVAIL equal to zero initially, which means that initially there is no process in the system and also no resource present.

4. The user inputs the number of processes and the number of resources as per his choice; firstly the resources are allocated as per the FCFS scheduling criteria fulfilling the demand of requested resources by processes if they are available in the system. These processes are executed and results are retrieved with the release of held resources.

5. Secondly if the demand DEM is less than the number of available resources AVAIL, 2 cases arise.

5(a) A process is holding some resources and request a resource which is held by some other process, then all the resources currently held are pre-empted, it may be restarted only it receives back its pre-empted resources as well the resources it demanded. It is reallocated resources and it follows the steps as the previous case 4.

5(b).The resources are neither available nor held by any waiting process, the process requesting for more resources has to wait and Consequently, some of its resources get pre-empted, if some other process requests resources of that type. This process could be restarted only it receives back its pre-empted resources as well as the resources it demanded. It is reallocated resources and it follows the steps as the previous case 5(a).

6. The released resources after process completion are maintained in the list .

7. The output is analysed as per the requirements and set of constraints, if it fulfils the criteria, the results are kept or it goes back to STEP 4.

8. Again the same procedure is followed for next allocation.

9. Outputs matrices are displayed for different matrices and it is indicated which process completes execution and its resources are sent back to resource table.

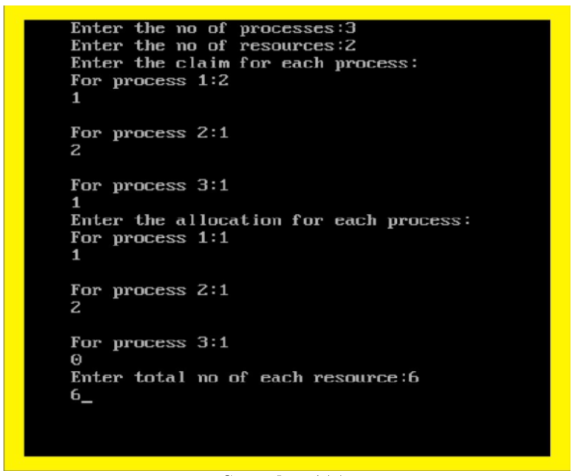
10.The output claim matrix and allocation matrix marks the end of simulation run.referenced

4 EXPERIMENT DETAILS AND RESULTS

4.1 Simulation test bench

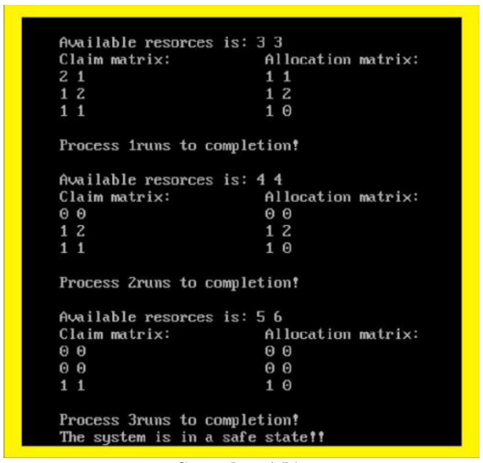
In our proposed model, we have done coding using C platform to have an better understanding of the various steps of the program.The Processor used is AMD A6-3420M APU.Operating system used is 64-bit Windows 7 with 4 GB RAM.We have a constraint for the maximum number of processes, It must be less than or equal to 10.And a constraint for the maximum number of resources, it must be less than or equal to 6.And this program code had been run in the main lab of the University as well as on our personal systems.

4.2 Snapshots



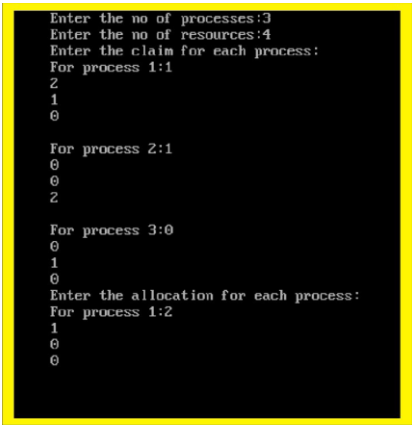
```
Enter the no of processes:3
Enter the no of resources:2
Enter the claim for each process:
For process 1:2
1

For process 2:1
2

For process 3:1
1
Enter the allocation for each process:
For process 1:1
1

For process 2:1
2

For process 3:1
0
Enter total no of each resource:6
6_
```

Snapshot 1(a).



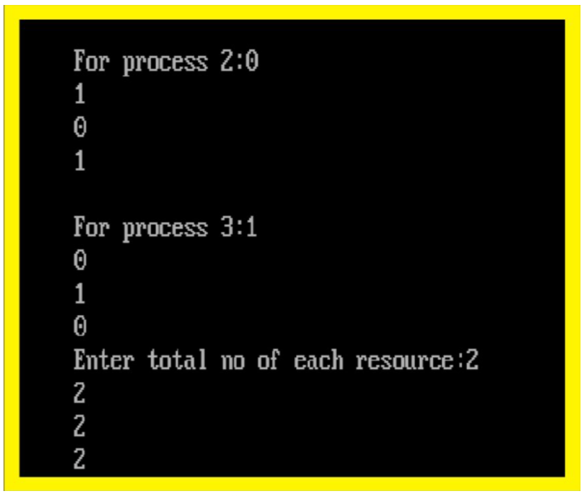
```
Available resorces is: 3 3
Claim matrix:             Allocation matrix:
2 1                       1 1
1 2                       1 2
1 1                       1 0

Process 1runs to completion!

Available resorces is: 4 4
Claim matrix:             Allocation matrix:
0 0                       0 0
1 2                       1 2
1 1                       1 0

Process 2runs to completion!

Available resorces is: 5 6
Claim matrix:             Allocation matrix:
0 0                       0 0
0 0                       0 0
1 1                       1 0

Process 3runs to completion!
The system is in a safe state!!
```

Snapshot 1(b).



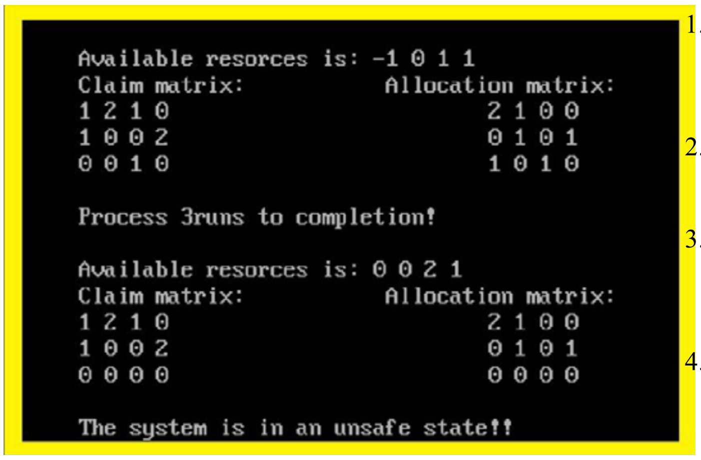
```
Enter the no of processes:3
Enter the no of resources:4
Enter the claim for each process:
For process 1:1
2
1
0

For process 2:1
0
0
2

For process 3:0
0
1
0
Enter the allocation for each process:
For process 1:2
1
0
0
```

Snapshot 2(a).



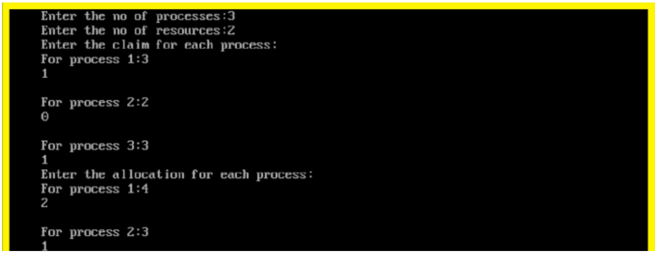
```
For process 2:0
1
0
1

For process 3:1
0
1
0
Enter total no of each resource:2
2
2
2
```

Snapshot 2(b).



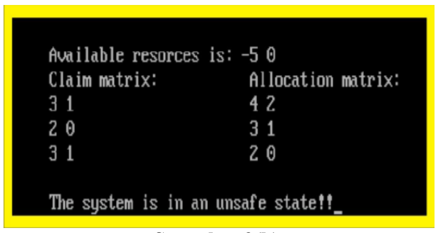
```
Available resorces is: -1 0 1 1
Claim matrix:           Allocation matrix:
1 2 1 0                 2 1 0 0
1 0 0 2                 0 1 0 1
0 0 1 0                 1 0 1 0

Process 3runs to completion!

Available resorces is: 0 0 2 1
Claim matrix:           Allocation matrix:
1 2 1 0                 2 1 0 0
1 0 0 2                 0 1 0 1
0 0 0 0                 0 0 0 0

The system is in an unsafe state!!
```

Snapshot 2(c).



```
Enter the no of processes:3
Enter the no of resources:2
Enter the claim for each process:
For process 1:3
1

For process 2:2
0

For process 3:3
1
Enter the allocation for each process:
For process 1:4
2

For process 2:3
1
```

Snapshot 3(a).



```
Available resorces is: -5 0
Claim matrix:        Allocation matrix:
3 1                  4 2
2 0                  3 1
3 1                  2 0

The system is in an unsafe state!!_
```

Snapshot 3(b).

5 Conclusion and Future Scope

Nowadays, everyone talks of the technologies like Grid and cloud Computing,. Every researcher aims working in these new technologies. So, considering the importance of a Grid environment, it is also necessary to think of the possible parameters that could affect a Grid environment, Deadlocks are one of those parameters and its hazards are very costly and time effective to recover. So, as It is quoted "Prevention is better than cure''. In this research paper, a new deadlock prevention algorithm has been offered with the emergence of grid computing. The main objective of paper is to prevent deadlock problem in Grid environment in order to preserve the data consistency and increase the throughput by maximizing the availability of resources and to ensure that all the resources available in the grid are effectively utilized. The work implements a simulation, and thus there is a need to validate the results obtained through hardware based system within the proposed indoor environment.

REFERENCES

1. U. Kapasi, W. Dally, S. Rixner, J. Owens, and B.Khailany,The Imagine stream processor Proc. International Conference of Computer Design, pg-282–288,2002
2. H.M.Deite, An Introduction to Operating Systems.Addision-Wesley Company, Second Edition, 199003- 8575-6/04, IEEE.
3. A.D.Kshemkalyani and M. Singhal,.A One-Phase Algorithm to Detect Distributed Deadlocks in Replicated Databases. IEEE Transaction on Knowledge and Engineering, Vol. 11, No.6.,1999.
4. ZhiWu Li, NaiQiWu, and MengChu Zhou.,Deadlock Control of Automated Manufacturing Systems Based on Petri Nets"—A iterature Review‖. IEEE transactions on systems, man, and cybernetics—part c: applications and reviews, Digital Object Identifier 10.1109/TSMCC.2011.2160626, IEEE.2011.
5. Nisha Sharma, Shivani, Saurabh Singh, Deadlock in Distributed Operating System. International Journal of Research in Information Technolog(IJRIT),pg 28-33,2013
6. Victor Fay Wolfe, Susan Davidson & Insup Lee. Deadlock Prevention in the RTC Programming System for Distributed Real-Time Applications, IEEE,1993
7. S. Venkatesh, J. Smit., An evaluation of deadlockhandling strategies in semiconductor

cluster tools IEEE Trans.Semiconductor Manufacturing, vol 18, pp. 197- 201.,2005

8. Lin Lou1 et.al.,An Effective Deadlock Prevention Mechanism for Distributed Transaction Management. Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing DOI 10.1109/IMIS.2011.109 IEEE Computer society.2011
9. Lei Gao, et.al, Resource Optimization and Deadlock Prevention while Generating Streaming Architectures from Ordinary Programs. NASA/ESA conference on adaptive hardware and systems IEEE.2011
10. Ajay D. Kshemkalyani , Mukesh Singhal ,.A One-Phase Algorithm to Detect Distributed Deadlocks in Replicated Databases, IEEE Transactions on Knowledge and Data Engineering v.11 n.6, p.880-895, 1999
11. Zhang Chuanfu Liu, A Deadlock Prevention Approach based on Atomic Transaction for Resource Co-allocation, Proceedings of the First International Conference on Semantics, Knowledge, and Grid (SKG 2005) IEEE.2006.