Framework for Preventing Deadlock : A Resource Co-allocation Issue in Grid Environment

Dr. Deepti Malhotra
Department of Computer Science and Information Technology
Central University of Jammu, Jammu, J&K, India

Abstract- In distributed environments, like Grid multiple resources are shared over the network in order to accomplish various tasks. However, to optimize the network performance and to complete the jobs successfully, these resources need to be used efficiently and effectively. Computational Grids have the potential for solving large-scale scientific problems using heterogeneous and geographically distributed resources. Due to the lack of centralized control and the dynamic nature, availability of the resource is very difficult. One problem that is critical to the effective utilization of computational Grids and gives a certain Quality of Service (QoS) for Grid users is the efficient co-allocation of jobs. Resource Co-allocation is fundamental to Grid computing. In the Grid computing community, the term co-allocation usually refers to the simultaneous access to resources hosted by autonomous providers. Resource Co-allocation involves the interaction of multiple entities, namely clients and resource providers. Multiple clients may ask for resources at the same time from the same providers. This situation may generate deadlocks if the resource providers use a locking procedure or if time out is associated with the locks. The aim of research paper is to design an efficient and effective framework to handle the distributed transactions in order to prevent deadlock and to ensure that all the resources available in the Grid are effectively utilized.

Keywords – Grid Computing, Resource Co-allocation, Deadlocks, ODP

I. INTRODUCTION

One of the promises of Grid Computing is to enable the execution of applications across multiple sites. Some of these applications require coordinated access to resources managed by autonomous entities. This coordinated access is known as resource co-allocation. There are two main classes of applications that require resource co-allocation: parallel applications with inter-process communication, and workflow applications. Parallel applications with inter-process communication require all resources to be available at the same time, whereas workflows constitute the execution of tasks with precedence constraints, i.e. resources have to be available in a certain order. Although both application classes require co-allocation, in the Grid computing community, the term co allocation usually refers to the simultaneous access to resources hosted by autonomous providers.

A number of distributed applications require simultaneous access to resources located in multiple administrative domains. In order to co-allocate resources in these environments, users need to reserve them in advance to meet their expected Quality of Service and enhance utility of their applications. Resource providers must frequently update their scheduling queues to reduce turnaround time of user requests and increase resource utilization. However, as all sub-requests of a co-allocation request must start at the same time, any modification in a single site may affect the requests of other sites. Due to the management complexity of resource co-allocation requests, their current usage is based on static advance reservations, which results in a low system utilization. There are a number of applications, in both academic and commercial environments, that require resource co-allocation. Some examples are (i) applications that require computing power that is not available in a single site; (ii) applications that require different resource types that are not available in a single site; (iii) users who need to speed up the execution of their applications; and (iv) redundancy of resources to improve fault tolerance during an execution. In resource management policies, the large scale parallel applications require multiple supercomputers to deal with problems such as splitting the application according to resource availability, performing the rescheduling when there are updates in the scheduling queues and keeping synchronized the co-allocation requests in the different resource providers.

To take full advantage of the promising capabilities of Grid computing, good resources co-allocation schemas must be developed. Compared to the resources allocation in traditional distributed systems, resources co-allocation in the Grid must consider the characteristics of users and the nature of resources, making the design of a co-allocation system very complicated [1]. Sometimes, the needs of a single job may exceed the capacity available in

each of the subsystems making up a Grid, and so co-allocation [2] (i.e. the simultaneous access to resources of possibly multiple types in multiple locations managed by different resource managers or locals scheduler) may be required.

The two main reasons for executing applications on multiple sites are: (i) the lack of special resources in a single administrative domain, such as devices for generating data, visualization tools, and supercomputers; and (ii) reduce response time of parallel applications by increasing the number of resources [2]. However, there are other applications that require co-allocation. Conference and multimedia users engaged in activities, such as scientific research, education, commerce, and entertainment, require co-allocation of multiparty real-time communication channels [3].Data-intensive applications use co-allocation to collect data from multiple sources in parallel [4] In addition, increasing the number of resources is a requirement of large-scale applications demanding considerable amounts of memory, storage, and processing power.

The rest of the paper is organized as follows. Resource co-allocation issues are explained in section II. Section III shows some interesting results from the previous research on resource co-allocation protocols. Section IV provides the details of the proposed deadlock prevention. Section V provides the results of the experiment carried out using our own grid simulated environment. Finally Section VI concludes the paper by summarizing the contributions and future works.

II. RESOURCE CO-ALLOCATION ISSUES

Existing work on resource co-allocation have focused on four research problems: distributed transactions, fault tolerance, inter-site network overhead, and schedule optimization. Resource co-allocation involves the interaction of multiple entities, namely clients and resource providers. Multiple clients may ask for resources at the same time from the same providers. This situation may generate deadlocks if the resource providers use a locking procedure; or livelock if there is a timeout associated with the locks. Therefore, there has been research on protocols to handle distributed transactions in order to avoid deadlocks and livelocks, and minimize the number of messages during these transactions. 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.

Another common problem in the resource co-allocation field is that a failure in a single resource compromises the entire execution of an application that requires multiple resources at the same time. One approach to minimize this problem is defining a fault tolerance strategy that notifies applications of a problem with a resource.

III. BACKGROUND AND RELATED WORK

Resource co-allocation to tasks is an important problem for Grid computing systems. As an effective solution, agent coalition formation had become a research hotspot [5]. A resource allocation strategy via agent coalition formation for real-time, dynamic, time-bounded Grid computing systems is presented. A scalable, decentralized resource co-allocation protocol that can facilitate the dependable deployment of Internet applications is presented in [6-7]. In this research, the proposed order-based deadlock prevention protocol with parallel requests (ODP3) protocol ensures deadlock and live-lock freedom during the resource co-allocation process. At the same time, the protocol takes advantage of parallelism in making resource allocation requests in order to achieve increased efficiency. ODP3 protocol requires the use of parallel requests feature, a set of goal states associated to each job in which achieving one of the goals satisfies the condition to run the corresponding job.

In contrast to ODP3, ODP2 does not require using parallel requests .To prevent deadlock and to reduce the degree of starvation of resource allocation, this protocol was proposed in expense of some global assumption: distinct resources need to be globally ordered. This assumption makes this protocol neither scalable nor fully decentralized to some extent. This protocol simply applies a multiple requests feature to globally linear ordered resources [8]. The order-based deadlock prevention protocol (ODP2) requires each job (or the corresponding co-allocator) to secure its resources one by one in an increasing order according to the given global order. Once all resources are allocated the job starts running. This protocol is deadlock-free, starvation-free and livelock-free. The weak points in this protocol are: resource usability, scalability, and central point of failure - a central node is required to keep global order among resources.

In article [9], Qi Chao et al. proposed an improved ant colony system (ACS) based algorithm, which efficiently solve the deadlock problem of tasks that the interdependence between tasks fails to consider during the course of resource assignment and task scheduling based on the general heuristics algorithm.

Agent-based resources co-allocation in Grid computing [10] had been introduced where a set of co-allocators agents receive jobs from multiple Grid users and use some techniques to schedule the job to one or more resources agent. The objectives of introducing these co-allocation strategies are as follows: (i) improve users benefit by minimizing their job's execution time and waiting time; (ii) improve resources benefit by maximizing theirs utilization rate. We assume that the co-allocation model is non-preemptive, and all the jobs are independent.

IV. 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-*
   a) *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.*
   b) *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.

4.1 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-*
   a) *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.*
   b) *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.*
   c) *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.*

4.2 Flowchart

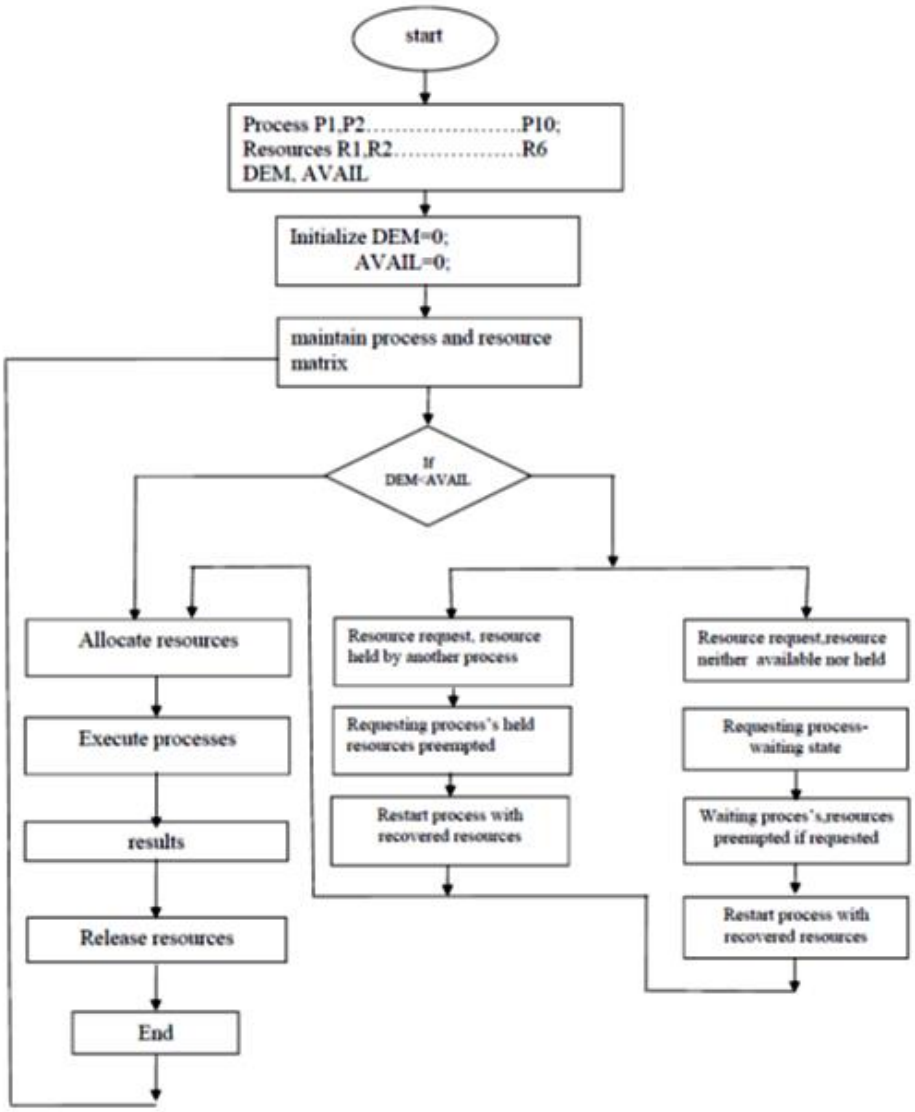


Figure1: Flowchart of Deadlock Prevention Algorithm

4.3 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.

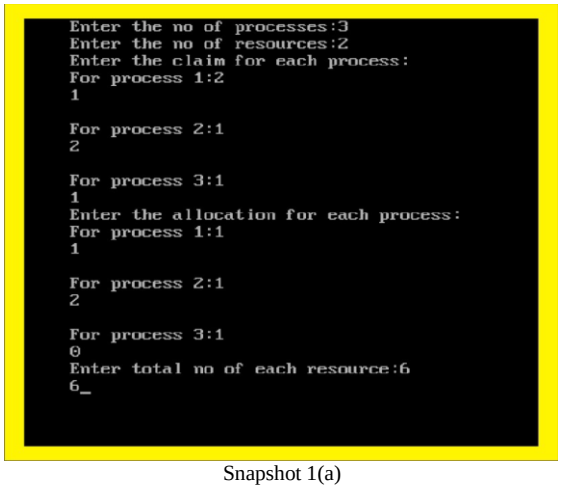
5) *Secondly if the demand DEM is less than the number of available resources AVAIL, 2 cases arise.*
   - 5(a) *A process is holding some resources and request a resource which is held by some other process, then all the resources currently held are pre-empted, it may be restarted only it receives back 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 analyzed 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.*

V. EXPERIMENT DETAILS AND RESULT

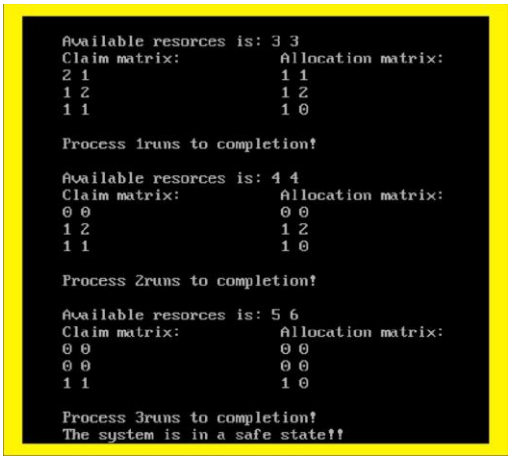
5.1 Simulation test bench

In our proposed model, we have done coding using C platform to have a 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.

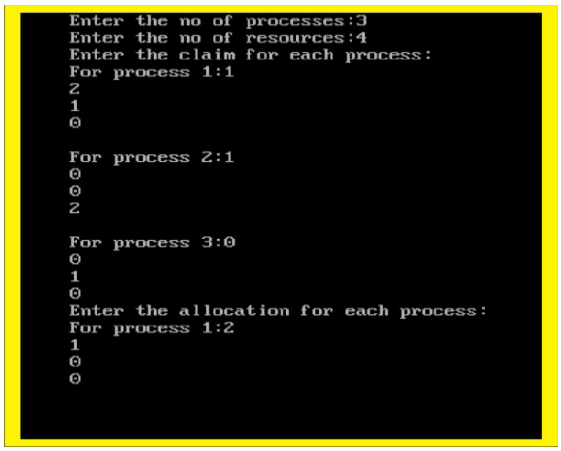
5.2 Snapshots



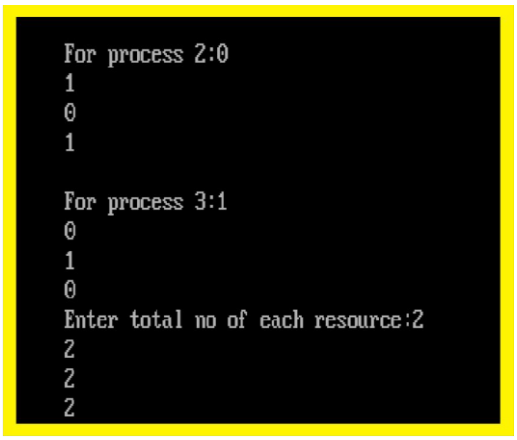
Snapshot 1(a)



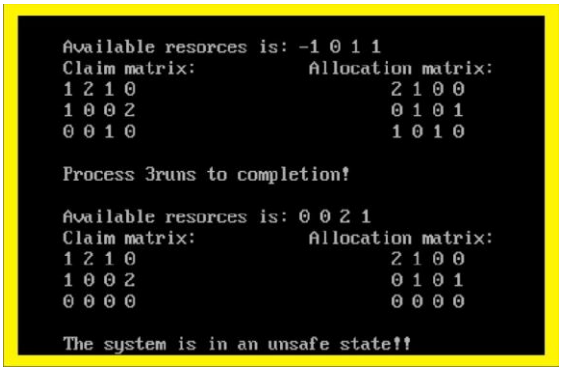
Snapshot 1(b)



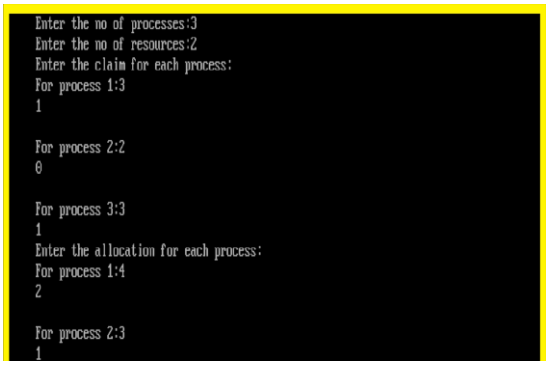
Snapshot 2(a)



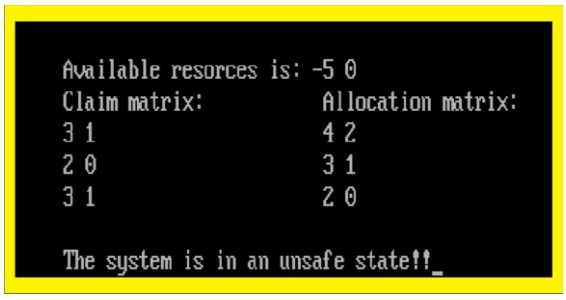
Snapshot 2(b)



Snapshot 2(c)



Snapshot 3(a)



Snapshot 3(b)

VI. Conclusion and Future Scope

Experience shows that the co-allocation of multiple resources is a challenging problem in Grid environments, due to application requirements for multiple resources and the inherent unreliability of the resources in question. Resource Co-allocation is fundamental to Grid computing. In the Grid computing community, the term co-allocation usually refers to the simultaneous access to resources hosted by autonomous providers. Resource Co-allocation involves the interaction of multiple entities, namely clients and resource providers. Multiple clients may ask for resources at the same time from the same providers. This situation may generate deadlocks if the resource providers use a locking procedure or if time out is associated with the locks. 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’’. A novel deadlock prevention algorithm with the emergence of Grid computing has been presented in this paper with the objective 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 proposed algorithm can subtly assign the

appropriate resources to tasks that exactly satisfy the needs of tasks. 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] H. Blanco, J. Lrida, F. Cores, and F. Guirado, "Multiple job co-allocation strategy for heterogeneous multi-cluster systems based on linear programming," The Journal of Supercomputing, pp. 1–9, 2011.

[2] K. Czajkowski, I. Foster, and C. Kesselman, "Resource co-allocationin computational grids," in HPDC '99: Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing.IEEE, Computer Society, 1999, p. 37

[3] D. Ferrari, A. Gupta, and G. Ventre, "Distributed advance reservation of real-time connections",ACM/Springer Verlag Journal on Multimedia Systems, 5(3), 1997.

[4] Rajkumar Buyya, Marco A. S. Netto ,"Resource Co-Allocation in Grid Computing Environments",IGI Global,2012.

[5] H.-J. Zhang, Q.-H. Li, and Y.-L. Ruan, "Resource coallocation via agent-based coalition formation in computational grids", International Conference on Machine learning and Cybernetics, 3:1936–1940, November 2003.

[6] J. Park, "A scalable protocol for deadlock and livelock free co-allocation of resources in internet computing", Proceedings of the Symposium on Applications and the Internet, 27(31):66–73, January 2003.

[7] J. Park, "A deadlock and livelock free protocol for decentralized internet resource coallocation", IEEE Transactions on Systems, Man, and Cybernetics, 34(1):123–131, January 2004

[8] Driss Azougagh,Jung-Lok Yu, Jin-Soo Kim, Seung-Ryoul Maeng , "Resource Co-Allocation : A Complementary Technique that Enhances Performance in Grid Computing Environment",IEEE, Proceedings of the 2005 11th International Conference on Parallel and Distributed Systems (ICPADS'05) ,2005.

[9] Qi Chao, Zhang Jing, and Li Jun-Huai, "ACS-based resource assignment and task scheduling in grid," Journal of Southeast University (English Edition), Vol. 23, No. 3, pp.451-454, Sept. 2007.

[10] Sid Ahmed MAKHLOUF, Belabbas YAGOUBI , "Distributed Resources Co-allocation in Grid Computing" , 978-1-4244-8611-3/10/$26.00 ©2010 IEEE,p 244-249.