SAJET

South Asian Journal of Engineering and Technology Vol.2, No.15 (2016) 1–11

ISSN No: 2454-9614

Dynamic Load Balancing on Deadline Task in Gridsim on Computational Grid

T. Tharani[a], R. Chellamani[a*]

[a)] *Department Of Computer Science and Engineering, Thiagarajar college of Engineering, Madurai, Tamilnadu, India.*
[b)] *Department Of Computer Science and Engineering, Thiagarajar college of Engineering, Madurai, Tamilnadu, India.*

*Corresponding Author: T. Tharani
E-mail: tharanit20@gmail.com,

Received: 01/11/2015, Revised: 20/12/2015 and Accepted: 25/02/2016

Abstract

Grid computing is a collection of computer resources from multiple locations to reach a common goal. The main concept behind gird computing is to make numerous autonomous machines that may be in different physical locations, act like they are a single virtual machine. Load balancing in a Grid is a hot research issue which affects every aspect of the Grid, including service selection and task execution. Thus, it is necessary and significant to solve the load balancing problem in a Grid. This paper describes the problem of scheduling and load balancing in a grid environment. A Dynamic load balancing algorithm is proposed which combines the strong points of neighbour based and cluster based load balancing techniques. This algorithm estimates system parameters such as resource processing capacity, load on each resource and transfer delay for scheduling and load balancing. A set of simulation experiments show that the proposed algorithm provides significant performance over existing ones.

Keywords: Load Balancing, Computational Grid , Gridsim , job scheduling , Resource state , Response Time , communication cost.

Reviewed by* **ICETSET'16 *organizing committee*

1. Introduction

Load balancing in a Grid is a hot research issue which affects every aspect of the Grid, including service selection and task execution. In our scenario, first, resources check their state and make a request to the Grid Broker according to the change of load state. Then, the Grid Broker assigns Gridlets **[4]** between resources and scheduling for load balancing **[2]** under the request. One important advantage grid computing is the provision of resources to the users that are locally unavailable. Load balancing can be defined by the following policies **[3].**

(1) The information policy specifies what workload information is to be collected, when it is to be collected, and

from where.

(2) The triggering policy determines the appropriate time at which to start a load balancing operation.

(3) The resource type policy classifies a resource as a server or a receiver of tasks according to its availability status.

(4) The location policy uses the results of the resource type policy to find a suitable partner for a resource provider or a resource receiver.

(5) The selection policy defines the tasks that should be migrated from overloaded resources (source) to the idlest resources (receiver).

Grid systems are classified into two categories: compute and data grids. In compute grids, the main resource that is being managed by the resource management system is compute cycles (i.e. processors) while in data grids the focus is to manage data distributed over geographical locations. The type of grid system it is deployed in affects the architecture and the services provided by the resource management system.

Load Balancing is one of the most important factors which affect the overall performance of application. Collection of load information from the other resources makes the resources to take load balancing decision. Existing techniques increases communication cost while collecting load information about resources and also cause negative impact on scalability. In this paper, scheduling and balancing application load for a computational grid is done by taking into account grid architecture, computer heterogeneity and communication delay.

A typical distributed system will have a number of interconnected resources which can work independently or in cooperation with each other. Each resource has owner workload, which represents an amount of work to be performed and every one may have a different processing capability. To minimize the time needed to perform all tasks, the workload has to be evenly distributed over all resources based on their processing speed. The essential objective of a load balancing consists primarily in optimizing the average response time of applications, which often means maintaining the workload proportionally equivalent on the whole resources of a system.

In this paper, scheduling and balancing application load for a computational grid is done by taking into account grid architecture, computer heterogeneity and communication delay.

1.1 Background

Load balancing must perform a critical role in enhancing the utilization of each resource and shortening the execution of time. In addition, policy and method of load balancing will directly affect the performance of the grid system **[4]**. Load-balancing algorithms can be roughly categorized as centralized, decentralized, or hierarchical in terms of location where the load-balancing decisions are made **[5].**

Static load balancing algorithms allocate the tasks of a parallel program to workstations based on either the load at the time nodes are allocated to some task, or based on an average load of our workstation cluster. The decisions related to load balance are made at compile time when resource requirements are estimated. It provides simplicity in terms of both implementation as well as overhead, since there is no need to constantly monitor the workstations for performance statistics. Static algorithms only work well when there is not much variation in the

load on the workstations. *Dynamic* load balancing algorithms make changes to the distribution of work among workstations at run-time; they use current or recent load information when making distribution decisions. Multicomputers with dynamic load balancing allocate/reallocate resources at runtime based on no a priori task information, which may determine when and whose tasks can be migrated. However, this comes at the additional cost of collecting and maintaining load information, so it is important to keep these overheads within reasonable limits

A load balancer is classified as *centralized*; *decentralized* and *hierarchal* which define where load balancing decisions are made. In *centralized* approach, one resource in a distributed system acts as the central controller. It has a global view of the load information in the system and decides how to allocate jobs to other resources. When the system size increases, the global knowledge of the system attributes is prohibitive due to the communication overhead produced and the central controller becomes a system bottleneck and single point of failure. In *decentralized* approach, all resources in the distributed system are involved in making load balancing decisions. Since load balancing decisions are distributed, it is costly to let each resource obtain the dynamic state information of whole system. Hence, most algorithms only use partial information stored in the local resource to make a sub-optimal decision. In a *hierarchical* model, the schedulers are organized in a hierarchy. High-level resource entities are scheduled at higher levels and lower level smaller sub-entities are scheduled at lower levels of the scheduler hierarchy. This model is a combination of centralized approach and decentralized approach.

Load balancing algorithms can be defined by their implementation of the following policies: *Information policy* specifies what workload information to be collected, when it is to be collected and from where. *Triggering policy* determines the appropriate period to start a load balancing operation. *Resource type policy* classifies a resource as server or receiver of tasks according to its availability status. *Location policy* uses the results of the resource type policy to find a suitable partner for a server or receiver. *Selection policy* defines the tasks that should be migrated from overloaded resources (source) to most idle resources (receiver).

The main objective of load balancing methods is to speed up the execution of applications on resources whose workload varies at run time in unpredictable way. Hence, it is significant to define metrics to measure the resource workload. Every dynamic load balancing method must estimate the timely workload information of each resource. Several load indices have been proposed in the literature, like *CPU queue length*, *average CPU queue length*, *CPU utilization*, etc. The success of a load balancing algorithm depends from stability of the number of messages (small overhead), support environment, low cost update of the workload, and short mean response time which is a significant measurement for a user. It is also essential to measure the communication cost induced by a load balancing operation.

A load balancing algorithm in which a computing node exchanges information and transfers jobs to its physical and/or logical neighbours is called *neighbour-based* load balancing method. Load balancing algorithms in which the computing nodes are partitioned into clusters based on network transfer delay are called as *cluster-based*

load balancing methods.

Scheduling algorithms can be separated into two types: *batch mode* and *on-line mode*. In *batch mode*, jobs are queued and collected into a set. The scheduling algorithm will start after a fixed time period. Batch mode heuristic algorithms are more appropriate for environments utilizing the same resource. In *on-line mode*, jobs are scheduled when they arrive. Since the grid environment is heterogeneous and speed of each processor various quickly, on-line mode heuristic scheduling is more appropriate for grid environment.

Scheduling algorithms can further be classified into: pre-emptive and non pre-emptive. In pre-emptive scheduling, an executing job can be pre-empted by another job. In a non pre-emptive scheduling, running jobs cannot be interrupted during their execution. Our work is focused on non pre-emptive, on-line mode heuristic scheduling of a periodic and independent job.

The rest of the paper are organised as follows: Section 2 reviews the Literature Survey. Section 3 describes the proposed load balancing model architecture. Section 4 presents the load balancing algorithms. Section 5 discusses the experimental environments and simulation setup. Section 6 gives simulation results, conclusion. Section 7 gives Acknowledgements and References.

2. Literature Survey

Keqin Li **[10]** proposed an optimal load balancing in a non dedicated cluster with heterogeneous servers. The optimization problem is solved for three queuing disciplines, namely dedicated applications without priorities, prioritized dedicated applications without preemption, and prioritized dedicated applications with preemption. Malarvizhi Nandagopal et al. **[5]** proposed a sender-initiated decentralized dynamic load balancing scheme for multi-cluster computational grid environment (SI-DDLB). D. Grosu et al. **[8]** proposed a non-cooperative load balancing game for distributed systems, but did not consider the communication delays in a grid environment. U. Karthick Kumar **[3]** proposed a dynamic load balancing algorithm for fair scheduling. Tasks are scheduled using fair completion time and rescheduled using mean waiting time of each task to obtain load balance.

Malarvizhi Nandhagopal and Rhymend V. Uthariaraj **[7]** addressed the problem of load balancing and scheduling in a grid resources where computational resources are dispersed in a different administrative domains. It addresses the problem of load balancing using min load and min cost policies while scheduling jobs to multi cluster environment. It considers both network load and communication cost for scheduling jobs to resources in different clusters. Three step strategies are used to determine a resource for an arriving job. Stylianos Zikos and Helen D. Karatza **[4]** proposes a load balancing and site allocation scheduling of unpredictable jobs in two level heterogeneous grid architecture (GS, LS). Three scheduling policies (Basic hybrid, PAD, FZF) at grid level which utilize site load information are examined. Shortest queue policy is used at the resource level for allocating jobs to PEs. These policies utilize dynamic site load information to share the load while communication overhead due to

information exchange is taken into account.

3. Load Balancing Model

3.1 Grid Model

As topological point of view, the grid consists of a set *C* of *n* resources *C* = *c*1, *c*2,....*cn* with set *G* of *n* grid schedulers *G* = *g*1, *g*2,….*gn* . Each grid scheduler *gi* runs on the resource *ci*. The resources are connected via different communication links which are viewed as internet links and modeled according to **[13]** and **[14]**. For simulation without loss of generality and to emphasize the basic ideas of the algorithms, the assumption is that each resource consists of one machine and each resource consists of different number of processors. Each resource has different processing capacity. The Grid Client generates jobs to be executed by the processors. Grid Clients send their jobs to grid scheduler for processing.

Each components of resource in the grid system can represent one or a combination of the following.

- *Scheduler*: This receives jobs from a set of grid clients and assigns them to the processors in the grid system.
- *Computational nodes*: This executes and processes jobs sent to it.
- *Load balancer*: This interacts with the scheduler and provides load control among computational jobs.
- *Dispatcher*: The dispatcher is responsible for dispatching job among processors.
- *GIS*: The grid information server is responsible for collecting and maintaining details of CPU utilization, processing capacity and load information among the resources.

3.2 Simulation of scheduling on Gridsim

A Grid simulation environment needs to abstract all the entities and the time dependent interactions in the real system. The layers of gridsim are explaining in detail. Grid Broker is the top manager of the grid environment which is responsible for maintaining the overall grid activities of scheduling and rescheduling the resources. The grid broker gets the information of the load from grid resources and sends the gridlet**[14]** to the resources for optimized scheduling

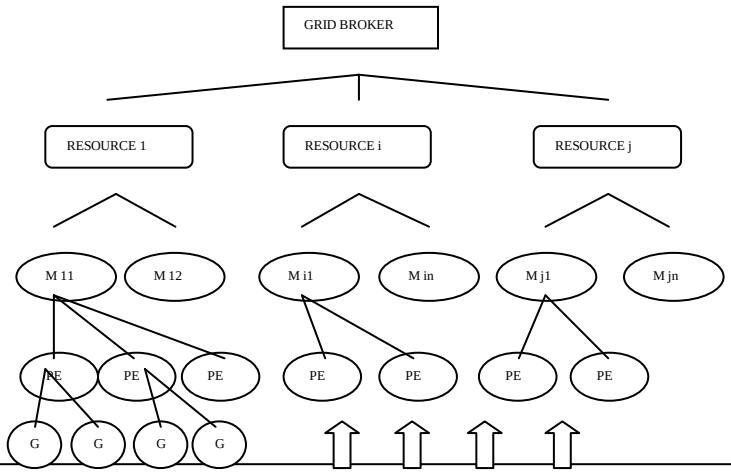

Fig 1 The structure of the Grid in GridSim.

Grid Resource is next to the Grid Broker in the hierarchy. It is responsible for maintaining the scheduling and load balancing in its machine. It sends an event to the grid Broker if it is overloaded. Machine is a processing entity (PE) manager. It is responsible for task scheduling and load balancing of its PEs. The Gridsim resource simulator uses an internal event to simulate the execution and allocation of PEs to gridlet jobs. When a job arrives, space shared systems start its execution immediately if there is a free PE otherwise it is queued. During the gridlet assignment, the job processing time is determined and the event is scheduled for delivery at the end of execution time. Whenever a gridlet finishes, an internal event is scheduled for delivery to signify the completion of scheduled gridlet job. GridSim 5.01 **[15]**, Gridlet, Grid Resource, and Machine are defined as follows.

- Gridlet (int gridletID, double gridlet Length, long gridlet File Size, long gridlet Output Size, Boolean record);
- Machine (int Machine ID, int numPE, int rating PE);
- Grid Resource(String name, double baud rate, long seed, Resource Characteristics resource, double peakLoad,
- Double off Peak Load, double relative Holiday Load,
- Linked List weekends, Linked List holidays);

Those parameters explaining can be found on the website **[16].**In our paper; we add two new parameters, i.e., deadline t to Gridlet and load level l to Grid Resource. We use Gridlet, Machine, and Grid Resource to illustrate the entity that would be used in Grid load scheduling, and we use deadline t to illustrate the request of a Gridlet. The resource simulator then frees the PEs allocated **[17]** to it and check if there are any jobs waiting in the queue. Then the resource simulator selects a suitable job depending upon the policy and assign to the PE which is free. The selection policy can be FCFS (First Come First Serve), SJF (Shortest Job First), HPF (Highest Priority First), HRN (Highest Response Next).If newly arrived event happens to be an internal event whose tag number is the same as the mostly recent scheduled event, and then it is recognised as the Grid completion event. The gridlets in the submission queue, depending on the allocation policy gridsim selects a suitable gridlet from queue to assign it PE or to the suitable PE with more than one PE is free. The resource speed and the job execution time can be defined in terms of ratings of standard benchmark such as MIPS and SPEC; upon obtaining the resource contact details from grid information service Grid broker query the resource directly.

3.3 Application Model

For any cluster *ci* in *C*, there are jobs arriving at *ci*, the jobs submitted to the grid are computation intensive, independent, non pre-emptive and a periodic with no required order of execution. The jobs are of different sizes meaning each job has different execution time and data transmission time for completion. Each job has different input file size and output file size requirements. The jobs are hold by the queue waiting for execution. All jobs in the queue *Q*(*ci*) are prioritized by their arrival time. It is assumed that only one job will be executed on a resource at a

time while others are waiting in the queue.

4. Load Balancing Algorithms

4.1 Grid Environment Installation

The GridSim resource simulator uses internal events to simulate the execution and allocation of PEs shared to Gridlet jobs. When jobs arrive, time shared systems start their execution immediately and share resources among all jobs. Schedule an internal event to be delivered at the earliest completion time of smallest job in the execution set. Newly arrived event happens to be an internal event whose tag number is the same as the most recently scheduled event, and then it is recognised as a job completion event. Should be noted that gridlets sharing the same PEs would get an equal amount of PE share. The completed Gridlet is sent back to its originator (broker or user) and removed.

4.2 Scheduling on Unassigned and waiting Gridlet

When a new Gridlet(s) is submitted by a user or a Gridletlist(s) is resubmitted by Overloadlist, there are many selection methods for the scheduling. The FPLTF (fastest processor to largest task first) **[18, 19]** algorithm schedules tasks to resources according to the workload of tasks in the Grid system. The algorithm needs two main parameters. They are the CPU (or PE) speed of resources and the workload of tasks. The scheduler sorts the tasks and resources by their workload and CPU (or PE) speed then assigns the largest task to the fastest available resource. Min–min **[20]** sets the tasks which can be completed earliest with the highest priority. The main idea of min–min is that it assigns tasks to the resources which can execute tasks in the fastest speed. Max–min **[20]** sets the tasks which have the maximum earliest completion time and the highest priority. Max–min overlaps the long running time task and short running time one.

Simulations show the significant by deliver the proposed technique compared to other approaches such as centralised load balancing and without load balancing. The thread concept is applied to perform the scheduling process in parallel. The assignment of Gridlet towards the resource by using roundrobin scheduling which is performed in parallel using threads. Round robin scheduling algorithm is employed, time slices are assigned to each process in equal portions and in circular order, handling all processes without priority .Roundrobin scheduling is simple, easy to implement and starvation free. In the absence of time sharing, if the quantam were large relative to the size of job, a process that produced large jobs would be favoured over there process. In order to schedule processes fairly, a round robin scheduler generally employs timesharing, giving each job a time slot or quantum to interrupting the job if it is not completed by then.

4.3 Current Load and Capacity of Resource calculation

Since every resource must manage itself, it cannot give all its processing abilities to the gridlet.Suppose that the load of one resource l. current load and the gridlet G is assigned to resource r with the deadline in t seconds. l.current load is the percentage of resource calculation ability that is given to gridlets. In fact, l.current load is the minimum of load l may be more than the result .Taking deadline t into account can extend the gridlet**[4].**The current

load of each resource is calculated using the formula below

l.currentload = l.currentload + g.gridletlength / t / resource capacity

Initially the current load is set to zero and calculate the load of the resource during the each assignment of the gridlet to the resource. By assumption that the resource l have a machine list as[machine 1…machine temp,…machine temp] and the machine temp can be illustrated as the machine temp(int machine idtemp, int num PEtemp, int rating PEtemp).

The number of PEs and the rating of PEs are represented as rating PE. Rating can be expressed as SPEC or MIPS. The capacity of the resource is calculated based on the number of processing entities currently active under the GridBroker. Based on the rate at which the resource executes its gridlet and result produced to the gridlet at the end of simulation. The capacity of the resource is calculated using the below formula as,

$$\text{Resource.capacity} = \sum \text{numPE(temp)n} \times \text{ratingPE(temp)}$$

Multitasking and multiprocessing systems allow currently running tasks to share system resources such as processor, memory, storage, I/O and network by scheduling their use for very short time intervals. A detailed simulation of scheduling tasks in the real system would be complex and time consuming. Hence, in Gridsim abstract these physical entities and simulate their behaviour using process oriented, discrete event "interrupts" with time interval as large as the time required for the completion of a smallest remaining job. Need to create a Gridsim user entity that creates and interacts with the resource broker scheduling entity to coordinate execution experiment. These simulated resources resemble the WWG tested resources used in processing a parameter sweep application using the Nimrod G broker. The brokers need to translate it into the GIS per MI (million instructions) for each resource.

4.4 Instantaneous Job Migration

In order to avoid the resource *ci* from getting overloaded, instantaneous job migration is used. The following describes the procedure for instantaneous job migration,

Sourceload = current load of the resource ci
Sort LNSet in ascending order based on load
If (sourceload > 0.97)
Migrate jobs to the resource ck in LNSeti having minimum load
Update load value of ck
End if

5. Simulation

The measuring of the make span has been widely used in evaluating the performance of a Grid **[19].** The make span is the ''total application execution time'', which is measured from the time the first job is sent to the Grid until the last job comes out of the Grid **[10].** In our simulation, we randomly generate Gridlets and topologies, although every simulation experiment yields roughly the same result. Make span is used to test the performance of

our algorithms in order to take into account realistic conditions. Some Gridlets were assigned to resource; due to the offline aspect of the resource or other reasons, the Gridlets may not be executed before their deadline. The number of Gridlets that cannot be finished on time is also selected as a standard for criteria for different schemes. Some Gridlets cannot be finished at the first resource scheduling, but can be scheduled again as its request. The sum of resubmitted time is another standard for our test. All the algorithms are simulated based on GridSim 5.0 [18]. The same configuration of resources is used in our simulations. We use Windows XP on an Intel Core (2.66 and 2.66 GHz), with 2048 MB of RAM and 360 GB of hard disk.

Table.1. Simulation parameters for Resource characteristics and scheduling

Simulation Parameters	**Value**
Number Of Resource	10
Number Of Machines per Resource	1
Number of PEs per machine	1-4
Processing Capacity of each PE	50-100 MIPS
Number of jobs	100-1000
Job Length	0-5000 MI
Input File Size	100 + (10-40%) MB
Output File Size	250 + (10-40%) MB

5.1 Number of Jobs versus Mean Response Time

Mean Response Time: Response time *rj* of job *j* is the time period from the job arrival to the completion time of the job i.e., the time spent in the resource queue plus the job service (execution) time. The mean response time *RT*,

$$RT = 1/N \sum_{j=1}^{N} r_j$$

The performance of the proposed work is compared with the non migration algorithm by varying the number of jobs. The system load is varied by varying the number of jobs submitted. The higher the load the higher the mean response time of both algorithms. By comparing these two algorithms, it is found that there is a considerable variation in the mean response time when the number of jobs increases.

Fig 2 Reliability Cost VS Span (ή)

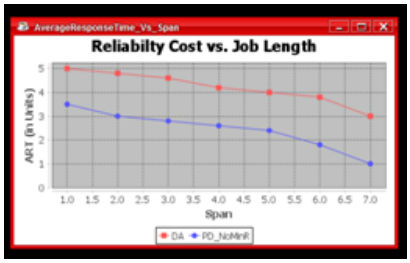

Fig 3 Reliability Cost VS Job Length

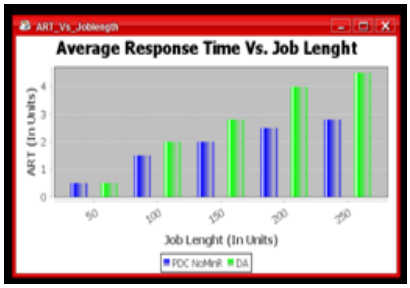

Fig 4 Average Response Time VS Job Length

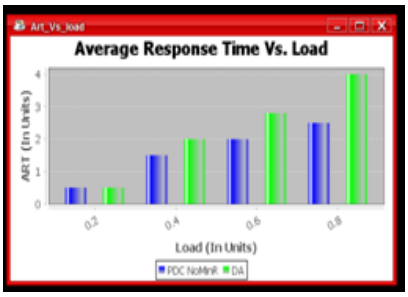

Fig 5 Average Response Time VS Load

6. Conclusion

In this paper, we propose a dynamic, distributed load balancing approach for a Grid, wherein resources and the Grid Broker participate in the load balancing operations. It provides an elastic mechanism for load balancing. Simulations prove that it can enhance the tasks finished rate and reduce the resubmitted time. In our future work, more characters, including the bandwidth, the resource processing ability, the requirement of Gridlet, will be taken into consideration.

In this paper a highly decentralized, distributed and scalable algorithm for scheduling and balancing loads across the resource in a heterogeneous grid environment is presented. The proposed model takes into account the heterogeneity of computational and network resources. The objective is to minimize the response time of jobs

arrived at a grid resource for processing.

References

[1] L. YUN-Han, L. Seiven, C. Ruay-Shiung, Improving job scheduling algorithms in a grid environment, Future Generation Computer Systems 27 (2011) 991–998.
[2] S. Dhakal, M.M. Hayat, J.E. Pezoa, C. Yang, and D.A. Bader, "Dynamic Load Balancing in Distributed Systems in the Presence of Delays: A Regeneration Theory Approach," IEEE Trans. Parallel Distributed Systems, vol. 18, no. 4, pp. 485-497, Apr. 2007.
[3] H.D. Karatza, Job scheduling in heterogeneous distributed systems, Journal of systems and Software (1994) 203–212.
[4] K.-Q. Yan, S.-S. Wang, S.-C. Wang and C.-P. Chang, "Towards a Hybrid Load Balancing Policy in Grid Computing System," Expert Systems with Applications, vol. 36, pp. 12054-12064, 2009.
[5] P.K Suri , M. Singh, "An Efficient Decentralized Load Balancing Algorithm for Grid," IEEE 'l Advance Computing Conf., pp.10-13.
[6] Keqin Li, "Optimal load distribution in non dedicated heterogeneous cluster and grid computing environments", *Journal of System Architecture*, Vol. 54, No. 2, pp. 111-123, 2008.
[7] Malarvizhi Nandagopal and V. Rhymend Uthariaraj, "Decentralized Dynamic Load Balancing for Multi Cluster Grid Environment", *Advanced Computing, First International Conference on Compt Science and Information Technology*, Vol. 133, pp. 149 - 160, 2011.
[8] D. Grosu and Anthony T. Chronopoulus, "Non Cooperative Load Balancing in Distributed Systems", *Journal of Parallel and Distributed Computing*, Vol. 65, No. 9, pp. 1022-1034, 2005.
[9] U. Karthick Kumar, "A Dynamic Load Balancing Algorithm in Computational Grid using Fair Scheduling", *International Journal of Computer Science*, Vol. 8, No. 5, pp. 123 - 129, 2011.
[10] Malarvizhi Nandagopal and V. Rhymend Uthariaraj, "Hierarchical Status Information Exchange Scheduling and Load Balancing for Computational Grid Environments", *International Journal of Computer Science and Network Security*, Vol. 10, No. 2, pp. 177 - 185, 2010.
[11] Stylianos Zikos and Helen D. Karatza, "Communication cost effective scheduling policies of non clairvoyant jobs with load balancing in a grid", *Journal of Systems and Software*, Vol. 82, No. 12, pp. 2103 - 2116, 2009.
[12] H. Casanova and L. Marchal, "A Network Model for Simulation of Grid Application", LIP Research Report, pp. 1 – 39, 2002.
[13] H.D. Karatza, Job scheduling in heterogeneous distributed systems, Journal of systems and Software (1994) 203–212.
[14] A. Legrand, L. Marchal and H. Casanova, "Scheduling Distributed Applications: The SimGrid Simulation Framework", *Proceedings of Third IEEE/ACM International Symposium on Cluster Computing and the Grid*, pp. 138 - 145, 2003.
[15] http://www.buyya.com/GridSim/ [visit:2011-1-27].
[16] L. YUN-Han, L. Seiven, C. Ruay-Shiung, Improving job scheduling algorithms in a grid environment, Future Generation Computer Systems 27 (2011) 991–998.
[17] D. Saha, D. Menasce, S. Porto, Static and dynamic processor scheduling disciplines in heterogeneous parallel architectures, Journal of Parallel and Distributed Computing 28 (1) (1995) 1–18.
[18] D. Paranhos, W. Cirne, F. Brasileiro, Trading cycles for information: using replication to schedule bag-of-tasks applications on computational Grids, in: International Conference on Parallel and Distributed Computing (Euro- Par), in: Lecture Notes in Computer Science, vol. 2790, 2003, pp. 169–180.
[19] M. Maheswaran, S. Ali, H.J. Siegel, D. Hensgen, R. Freund, Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing system, Journal of Parallel and Distributed Computing 59 (1999) 107–131.