DOI: 10.21917/ijsc.2013.0088

A DECENTRALIZED DYNAMIC LOAD BALANCING FOR COMPUTATIONAL GRID ENVIRONMENTS

R. Chellamani[1] and R. Sivaranjani[2]
Department of Computer Science and Engineering, Thiagarajar College of Engineering, India
E-mail: [1]rcmcse@tce.edu and [2]sivaranjani1290@gmail.com

Abstract
With the rapid development of high-speed wide-area networks and powerful yet low-cost computational resources, grid computing has emerged as an attractive computing paradigm. The computational grid is a new parallel and distributed computing paradigm that provides resources for large scientific computing applications. The main techniques that are most suitable to cope with the dynamic nature of the grid are the effective utilization of grid resources and the distribution of application load among multiple resources in a grid environment. This paper addresses the problem of scheduling and load balancing in a grid environment. A Decentralized Dynamic load balancing algorithm is proposed which combines the strong points of neighbor 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:
Grid Computing, Load balancing, Scheduling, Response Time, Job Migration

1. INTRODUCTION

Grid computing is a model of distributed computing that uses geographically and administratively disparate resources [1]. In Grid computing, individual users can access computers and data, transparently, without having to consider location, operating system, account administration, and other details. In Grid computing, the details are abstracted, and the resources are virtualized. Grid Computing has emerged as a new and important field and can be visualized as an enhanced form of Distributed Computing. Grid computing is the next generation IT infrastructure that promises to transform the way organizations and individuals compute, communicate and collaborate. The goal of Grid computing is to create the illusion of a simple but large and powerful self-managing virtual computer out of a large collection of connected heterogeneous systems sharing various combinations of resources. Sharing in a Grid is not just a simple sharing of files but of hardware, software, data, and other resources. Thus a complex yet secure sharing is at the heart of the Grid.

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.

A typical distributed system will have a number of interconnected resources who 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.

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.

1.1 BACKGROUND

Load balancing is a technique to enhance resources, utilizing parallelism, exploiting throughput improvisation, and to cut response time through an appropriate distribution of the application. To minimize the decision time is one of the objectives for load balancing which has yet not been achieved. A load balancing feature can prove invaluable for handling occasional peak loads of activity in parts of a larger organization.

Load balancing algorithms can be classified into two categories: *static* or *dynamic*. *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

There are three major parameters which usually define the strategy a specific load balancing algorithm will employ. The

question of who makes the load balancing decision is answered based on whether a *sender-initiated* or *receiver-initiated* policy is employed. In *sender-initiated* policies, congested nodes attempt to move work to lightly-loaded nodes. In *receiver-initiated* policies, lightly-loaded nodes look for heavily-loaded nodes from which work may be received.

Global or *local* policies answer the question of what information will be used to make a load balancing decision. In *Global* policies, the load balancer uses the performance profiles of all available workstations. In *Local* policies workstations are partitioned into different groups. The choice of a global or local policy depends on the behaviour an application will exhibit. For global schemes, balanced load convergence is faster compared to a local scheme since all workstations are considered at the same time.

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 aperiodic and independent jobs.

The rest of the paper are organised as follows: Section 2 reviews the related work in issue. 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 and section 7 gives concluding remarks.

2. RELATED WORK

Yongsheng Hao et al. [2] proposed a dynamic, distributed load balancing scheme for a grid which provides deadline control of tasks. They proposed a new calculation method and classified resource into three types: overloaded, normally load and under loaded. Unassigned gridlet list used to store the new arriving gridlet and the unfinished gridlets coming from the resource when the execution fails. Finally proposed a novel based load balancing method such that resources and grid broker participates in load balancing.

U. Karthick Kumar [3] proposed a dynamic load balancing algorithm for fair scheduling. It addressed fairness issues using mean waiting time. Tasks are scheduled using fair completion time and rescheduled using mean waiting time of each task to obtain load balance.

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.

Malarvizhi Nandagopal et al. [5] proposed a sender-initiated decentralized dynamic load balancing scheme for multi-cluster computational grid environment (SI-DDLB).

Yajun Li et al. [6] addressed the problem of load balancing by hybrid approach (both average based and instantaneous

measures based) for sequential tasks under grid computing. A first come first served and a carefully designed genetic algorithm were selected as representatives of both classes to work together to achieve load balancing. To trigger GA into operation, sliding window technique was used.

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.

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.

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.

The game-theoretic approach proposed by Zomaya et al. [11] considers only individual response time as the objective and does not consider average response time.

When compared with the existing work, the main characteristics of the proposed strategy can be summarized as follows,

- It privileges a decentralized load balancing.
- To minimize the overhead involved in site state information exchange among resources is done through mutual information feedback.
- Jobs are computation intensive.
- Jobs are non pre-emptable which means that their execution on a resource cannot be suspended under completion.
- Jobs are independent which means that there is no communication between them.

3. SYSTEM MODEL

3.1 GRID MODEL

As topological point of view, the grid consists of a set C of n resources $C = c_1, c_2,....c_n$ with set G of n grid schedulers $G = g_1, g_2,....g_n$. Each grid scheduler g_i runs on the resource c_i. The resources are connected via different communication links which are viewed as internet links and modeled according to [12] and [13]. 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 APPLICATION MODEL

For any cluster $c_i \in C$, there are jobs arriving at c_i, the jobs submitted to the grid are computation intensive, independent, non preemptive and aperiodic 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 $Q(c_i)$ 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. METHODOLOGY

4.1 LOAD BALANCING MODEL

Each resource c maintains a set of neighbors $LNSet_i$ and partners $LPSet_i$ for scheduling and load balancing. Neighbors are formed in terms of transfer delay. For a resource c_i, c_j is considered as neighbor as long as the transfer delay between c_i and c_j is within α times that of the transfer delay between c_i and its nearest neighbor. It is found that α =1.375 yields good result.

$$\alpha = \frac{TD_{ij}}{TD_{inearest}} \quad (1)$$

Thus for each resource, other neighbor resources are sorted by transfer delay in ascending order. The first ranked resource is chosen as the nearest neighbor resource.

Partners are formed in terms of processing capacity. For a resource c_i, c_j is considered as partner if the processing capacity of c_j is greater than that of c_i. Thus for each resource, other partner resources are sorted by processing capacity in descending order. The first ranked resource is chosen as the fastest resource.

4.2 MUTUAL INFORMATION FEEDBACK

In order to minimize the overhead involved in collecting load information among resources, Mutual information feedback policy is used. Each resource c_i maintains only the state information of itself, its neighbors and partners by using state object O_i. This state object helps a resource to estimate the load and efficiency of other resources without having message

transfer. Each item $O_i[k]$ has a property list (load, time). $O_i[k]$.load denotes the load information of resource c_k and $O_i[k]$.time denotes c_k's local time at which the load status is reported.

When c_i transfers a job transfer request to one of its neighbor or partners c_k for processing, c_i appends the information of itself, its neighbors and partners to the job transfer request sent to c_k by piggybacking. c_k then, updates the state information (load and time values) in its state object by comparing the timestamps if the resource contained in the job transfer request belong to its neighbor or partners. Similarly while sending job acknowledge or completion reply sent from c_k to c_i, c_k appends the state information of itself, its neighbors and partners to the job transfer request sent to c_i by piggybacking so that c_i can update its state objects.

An advantage of mutual information feedback policy is that the load dissemination rate is directly proportional to job arrival rate. An increase in job arrival rate increases the load information exchange more frequently.

4.3 INSTANTANEOUS JOB MIGRATION

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

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

Fig.1. Pseudocode for Instantaneous Job Migration

5. EXPERIMENTAL RESULTS

The Simulations are performed on Gridsim which is a java-based discrete event simulation toolkit [14]. A set of experiments are conducted against Non Migration (NM) algorithm and Instantaneous Job Migration algorithm (IJM).

The following table represents the simulation parameters for resource characteristics and scheduling.

Table.1. Simulation parameters for Resource characteristics and scheduling

Simulation Parameters	Value
Number of resources	25
Number of machines per resource	1
Number of PEs per machine	1-4
Processing capacity of each PE	50-100 MIPS
Number of jobs	100-3000
Job length	0-50000 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 r_j 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*,

$$RT = \frac{1}{N}\sum_{j=1}^{N} r_j \quad (2)$$

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.

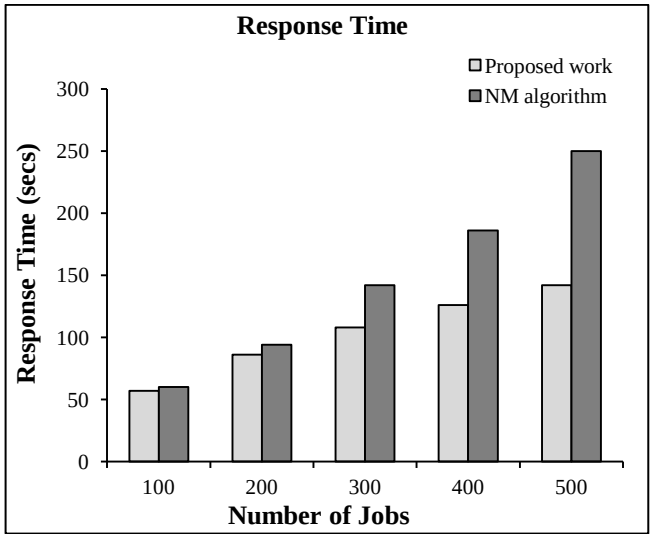

Fig.2. Number of jobs versus Mean Response time

5.2 NUMBER OF JOBS VERSUS SLOWDOWN

Slowdown: Slowdown S_j of a job j is the job's response time divided by the job's execution time. If e_j is the execution time of a job j, then the slowdown is defined as follows,

$$S_j = \frac{r_j}{e_j} \quad (3)$$

The average slowdown *SLD* is,

$$SLD = \frac{1}{N}\sum_{j=1}^{N} S_j \quad (4)$$

The performance of the proposed work is compared with the non migration algorithm by varying the number of jobs. It is found that there is a considerable variation in the slowdown when the number of jobs is more.

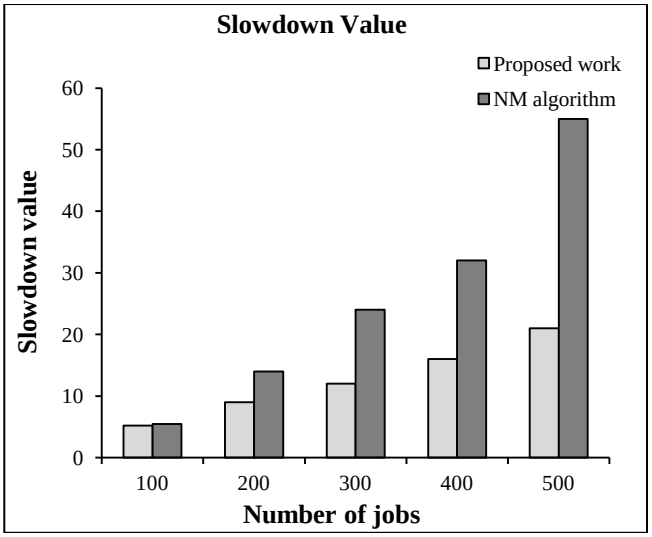

Fig.3. Number of jobs versus Slowdown value

6. CONCLUSION AND FUTURE WORK

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. To reduce communication overhead while collecting load information of other resources in the *LNSet* and *LPSet*, mutual information policy is used. For load balancing Load adjustment policy is applied which is compared with instantaneous job migration algorithm and non migration algorithm. It is found that this algorithm improves performance when compared with IJM and NM algorithm.

In this work, the jobs considered here are independent of one another, but different tasks may have same precedence constraints. It is planned to consider some fault tolerant mechanism to improve the reliability of our algorithm and also planned to explore the potential of these load balancing strategies by embedding this to real world grid computing environments.

ACKNOWLWEDGEMENTS

We thank the anonymous reviewers for their comments and recommendations, which have been helpful in improving the quality of this work.

REFERENCES

[1] Ian Foster and Carl Kesselman, "*The Grid: Blueprint for a new Computing Infrastructure*", 2nd Edition, Morgan Kauffmann Publishers, 2003.

[2] Yongsheng Hao, Guanfeng Liu and Na Wen, "An enhanced load balancing mechanism based on deadline control on GridSim", *Future Generation Computer Systems*, Vol. 28, No. 4, pp. 657 - 775, 2012.

[3] 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.

[4] 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.

[5] Malarvizhi Nandagopal and V. Rhymend Uthariaraj, "Decentralized Dynamic Load Balancing for Multi Cluster Grid Environment", *Advanced Computing, First International Conference on Computer Science and Information Technology*, Vol. 133, pp. 149 - 160, 2011.

[6] Yajun Li, Yuhang Yang, Maode Ma and Liang Zhou, "A hybrid load balancing strategy of sequential tasks for grid computing environments", *Future Generation Computer Systems*, Vol. 25, No. 8, pp. 819 - 828, 2009

[7] 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.

[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] C. Kandagatla, "Survey and taxonomy of grid resource management system", University of Texas, Available at http://www.cs.utexas.edu/users/, 2003.

[10] 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.

[11] Riky Subrata, Albert Y. Zomaya, Bjorn Landfeldt, "Game-Theoretic Approach for Load Balancing in Computational Grids", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 19, No. 1, pp. 66 - 76, 2008.

[12] H. Casanova and L. Marchal, "A Network Model for Simulation of Grid Application", LIP Research Report, pp. 1 – 39, 2002.

[13] 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.

[14] R. Buyya and M. Murshed, "Gridsim: a toolkit for the modelling and simulation of distributed resource management and scheduling for Grid computing", *Concurrency and Computation: Practice and Experience*, Vol. 14, No. 13-15, pp. 1175 - 1220, 2002.