Deadline Stringency based Job Scheduling in Computational Grid Environment

Sukalyan Goswami
Department of Computer Science & Engineering
Institute of Engineering & Management
Kolkata, India
sukalyan.goswami@gmail.com

Ajanta Das
Department of Computer Science & Engineering
Birla Institute of Technology, Mesra
Kolkata Campus, India
ajantadas@bitmesra.ac.in

Abstract* — *Computational grid, which came into existence in mid 1990s, represents a computing infrastructure, which is distributed in nature. Grid, these days, is extensively used in science and technology. As a result, computational grid has become a promising research field. It has become a highly productive platform for dynamic problem solving. Grid provides resource sharing among different organizations. Due to the provision of scalable resources, these days grid's usage is rapidly increasing in industry also. But, computational grid has few distinct requirements in comparison to those of traditional distributed high performance computing systems. To utilize such grid systems, resource allocation and utilization are key challenges to be dealt with. Henceforth load balancing is another common problem as because the load scenarios of individual resources are dynamic in nature. The objective of this paper is to enhance the efficiency of an already proposed NDFS algorithm to solve the existing problem of load balancing in computational grid. Comparisons of the performances of a few existing load balancing algorithms with proposed NDFS are also presented in this paper.

Keywords* - *Grid computing, load balancing, deadline stringency, resource allocation, resource utilization.

I. INTRODUCTION

Grids [1] were originally built to ensure resource and knowledge sharing within scientific community. A lot of progress has been made till now from the inception of grid infrastructure. The grid is a system which is capable to cater to large number of jobs. Grid computing is different from conventional high performance computing as because grids are more loosely coupled, heterogeneous and geographically dispersed. So, the computational grid [1], consists of distributed heterogeneous resources to solve different types of large-scale problems in engineering, science and commerce. Grids can be constructed from participating computing resources which may belong to geographically dispersed administrative domains.

Because of irregular task receiving patterns and uneven computational powers, different nodes in different grid sites will generally have unequal load patterns, some nodes may be under-utilized whereas some other may be highly overloaded. So, to exploit the full power of such grid systems, scheduling of jobs, allocation and management of resources are essential functions in grid to be dealt with. Moreover, balanced load has to be achieved across the grid to make the system more efficient.

The objective of this paper is to enhance the efficiency of an already proposed Nearest Deadline First Served (NDFS) [15] algorithm to solve the prevailing problem of dynamic load balancing in computational grid. In the proposed algorithm, the resource broker always receives job requirements from the clients as well as, simultaneously it collects runtime status information of resources for suitable allocation of resources to each client's request. Hence the efficiency of the algorithm is enhanced with parallelism in operation. Comparisons of the performances of a few existing load balancing algorithms with proposed NDFS are also presented in this paper. Section 2 discusses the related work done in load balancing applicable in grid computing. Section 3 presents the proposed algorithm and section 4 discusses the simulation results and comparisons with other algorithms. Section 5 concludes the paper.

II. RELATED WORKS

The computing nodes' utilization should be maximized and job execution time should be minimized if load balancing operation is carried out across the grid. The scheduling algorithms for load balancing may be classified into two categories: centralized and decentralized [10].

In centralized approach [10], a controller distributes workload among different grid sites. The central controller keeps the details of all the resources. The complexity of load distribution increases as the number of resources and number of jobs increase.

In the decentralized approach [9] information provided by the sites are crucial in performing the load balancing operation. Compared to centralized approach, this is better fault- tolerant and scalable.

Quite a few load balancing approaches for computational grid environment have already been put forward by researchers in the said arena. Among them the hybrid methods are commonly used. They perform balancing of workload in grid by using either genetic algorithm (GA) [10] or First-Come-First-Served (FCFS) [10] algorithm. FCFS is used for smaller number of jobs. But as the jobs in queue keep on increasing more and more, GA is used for performing workload balancing through sliding window. But shifting from one algorithm to another evidently makes the implementation more complex.

Workload balancing in grid may also be achieved by ant colony optimization (ACO) [10]. Initially, an ant finds an overloaded node by moving ahead at random, as because the ants are only allowed movement in search-max or search-min format. After finding overloaded node, the ant shifts to search-min mode to find an under loaded node. After searching is successful, the ant makes workload balanced between the adversely loaded nodes.

Cao [6] and his co-researchers [6] proposed usage of intelligent agents, which follow the mechanism of self-organization, like, ants to address the problem of workload balancing in computational grid environment. Here agents act as resources and they interact among themselves to exchange information among nodes resulting in job execution time reduction.

Three more algorithms, namely, Max–min, Sufferage and Min–min algorithms [10], if implemented in grid environment, can also perform workload balancing. The decision making criteria of job scheduling is different for these algorithms. The job with minimum completion time (MCT) is allocated first in Min–min algorithm, whereas in Max–min algorithm, the job with maximum completion time is chosen. The Sufferage algorithm allocates jobs based on the calculated sufferage value of each job. Sufferage value of a job is defined as the difference between best MCT and second best MCT. Another algorithm used for job scheduling is Fastest Processor to Largest Task First (FPLTF) [17] which schedules the largest job to the fastest processor available in the grid. Although Without Load Balancing (WLB) [7] and Load Balancing on Enhanced GridSim (LBEGS) [17] approaches do not consider the load scenarios of the resources in the grid, but these approaches are the easiest to implement in computational grid.

The load balancing mechanism in computational grid tries to achieve equal distribution of workload among the participating resources. Maximization of resource utilization and minimization of the total execution time are chosen as the most important parameters in this context. In order t achieve this, job scheduling plays the most important role. To deal with performance fluctuations of grid, a feasible scheduling algorithm must cater to such dynamic environment. On the other hand, a few important measures always diminish the impact of the above-mentioned problem in computational grid environment, such as resource reservation, Service Quality Agreement (SQA) negotiation and rescheduling of the jobs.

The discussed approaches have seldom given importance to the deadline stringency of the submitted jobs. This research concentrates on the prioritization of deadline of submitted jobs. There are two aspects of the proposed approach – *i) nearer the deadline of the submitted job, higher is its priority and ii) highest priority job gets allocated to least loaded processing nodes.* Hence, the proposed 'service oriented framework' described in this paper deals with the provision of the resources is capable of satisfying the Service Quality Agreement (SQA) of the task submitted by client.

III. Proposed Load Balancing Algorithm

To deal with these prevailing problems of computational grid environment, a 'Service Oriented Load Balancing Framework' is proposed in [14]. This framework follows a layered architecture, where the top most layer consisting of 'grid users' or clients submit computation or data intensive application to grid for execution. The middle layer representing the 'resource broker' is solely responsible for distribution of the jobs to the grid resources based on client's service quality requirements and details of available grid resources for further executions, and the bottom layer consists of pool of 'resources' including cluster, PCs, supercomputer etc. which executes the submitted jobs. These jobs are part of each client's application. Actually before scheduling the application to the specific resources, the broker might split each application into different mutually exclusive jobs.

The objectives of the proposed algorithm are to meet deadlines of the respective jobs submitted to the broker and enhancement of utilization of the resources. Since, the algorithm strives to meet deadline of job, hence it was named as *Nearest Deadline First Served* (NDFS). Balancing of load for each resource in computational grid directs to schedule jobs such that neither the resource will be heavily loaded nor it will be underutilized. Keeping this point in contention, scheduling of jobs based on nearest deadline eventually balances the load among the participating resources. In this paper, the proposed algorithm NDFS not only serves to the client rather it gives importance on job scheduling based on deadline stringency. Hence, from now onwards this research renames the already proposed algorithm as *Nearest Deadline First Scheduled* (NDFS).

In the proposed NDFS, broker plays the major role for balancing load. As per Kesler's model [5], resources can be various types and spread over the grid. Moreover, the clients can submit their jobs request at any time to the grid. As a result, the resource broker may always receive job requirements from the clients. Simultaneously it has to collect runtime status information of resources for suitable allocation of resources to each client's request. Resource broker in each grid site can communicate to more than one client at the same time. This approach speeds up the process by enhancing the efficiency of NDFS.

Figure 1 represents the following steps of the proposed enhanced algorithm in details assuming that maximum numbers of resources in computational grid environment are active.

Step 1(a) – *Broker collects job requirements from the clients:* The clients submit job requirements, like, processing power requirement, memory requirement and most importantly target time for completion of their job, to the resource broker.

Step 1(b) – *Broker collects runtime status of resources:* The resource broker of the grid collects runtime status or information from the dynamic resource pool. Broker keeps a sorted record of those which will be required during job scheduling.

Step 2 – *Preparation of Service Quality Agreement:* The job requirements submitted to the broker are 'service quality requirements', which, if only mutually agreed between broker and client, then only the SQA (Service Quality Agreement) is generated and a 'bipartite agreement' between the client and the broker is signed. The broker sorts the jobs based on their target deadlines.

Step 3 – *Scheduling of resources for allocation of jobs:* After signing of SQA, the broker allocates the highest priority job to the least loaded resource, provided that resource is capable of satisfying the SQA. In the process of scheduling resources, the broker consults two different sorted lists; one contains 'runtime information of resources' and the other having 'submitted jobs' requirements'. If two or multiple jobs are submitted to broker with same target deadline, then the one which is submitted first gets allocated to minimal utilized resource first.

In this proposed algorithm, while scheduling the jobs by the broker, importance is given mainly to the deadline of each job along with the resource utilization.

IV. EXECUTION RESULTS

This section presents execution results of the proposed algorithm along with other five existing algorithms. The computational grid environment is set up with four nodes having the following specifications as represented in TABLE I.

A. Simulation Environment

Java 7 is installed and Eclipse Juno version is used as IDE for all the resources. Thereafter, GridSim version 5.0 is also installed on top of these resources namely, Resource 1, Resource 2, Resource 3 and Resource 4.

Resource	Processor	RAM / HDD	OS
Resource 1	Intel core i3, 2350M, 2.30 GHz	4 GB / 750 GB	Windows 7 enterprise edition
Resource 2	Intel core i3, 2350M, 2.30 GHz	4 GB / 500 GB	Windows 7 enterprise edition
Resource 3	Intel core i3, 3210, 3.2 GHz	4 GB / 500 GB	Windows XP SP3
Resource 4	AMD AM3 FX4100, 3.6 GHz	4 GB / 1 TB	Windows 7 Ultimate edition

TABLE I Resource Specifications

The proposed algorithm is simulated with GridSim [16], which simulates computational grid environment efficiently. GridSim is capable of simulating resource entities having either single processor or multiprocessors. Resources can be modeled to be space-shared or time-shared systems. Moreover, GridSim can simulate the underlying networks connecting the resources. And it can also model entities for the broker, users, information service, network-based I/O and statistics. So, it possesses the property of abstraction of entities and their interactions and it also supports the generation of response functions which are time-dependent in nature and are defined by the user.

In this simulation, grid broker maintains the overall activities of the computational grid. It is the top manager of the grid environment and also performs scheduling of grid resources. A grid resource is represented by *resource entity*. Resource entities may possess different properties. Each resource, specifically machine (node), is a *processing entity (PE) manager*. It performs job scheduling and workload balancing of its PEs. All job related information is contained in *Gridlet*. It also contains execution management details such as, job length in MIPS (million instructions per second), the size of input and output files, the originator of the job and the job deadline time value. The execution time of job and the transportation time of input and output files among users and resources are determined by the above said parameters of GridSim. These are responsible for returning the results back to the job originator along with processed Gridlets.

Next section presents the simulation result.

B. Simulation Results

The simulation results of the proposed Nearest Deadline First Scheduled (NDFS) algorithm are compared with five existing algorithms, namely, Without Load Balancing (WLB), Max-Min, Min-Min, Load Balancing on Enhanced GridSim (LBEGS) and Fastest Processor to Largest Task First (FPLTF).

Three parameters have been compared for all the above said six algorithms when number of Gridlets is varied keeping PE constant at 200.

Figure 2 represents the comparison of number of communication overhead times. Performances of NDFS and FPLTF are best among above said algorithms.

Figure 3 represents the comparison of makespan, the application execution time. NDFS performs best among all.

And Figure 4Figure 4 presents the comparison of number of finished Gridlets, showing NDFS performing the best again. In all three cases, performance of WLB is worst showing the need of workload balancing in the grid.

V. CONCLUSION

The computational grid is the collections of computer resources, which are geographically disperse, providing cost-effective solutions for large-scale algorithmic processing problems. However, task scheduling, resource management and load balancing are most important grid services. To have balanced workload across grid, minimization of load difference needs to be achieved between the overloaded nodes and the underutilized nodes.

In this paper, performance of a few existing algorithms along with proposed NDFS is compared. The experimental results also prove that efficiency of the proposed NDFS is enhanced through introducing more parallelism (via resource broker) in operation.

However, the scenario of resource failure has not been considered in this paper. Therefore, this research is currently concentrating on a novel strategy in order to meet the target deadline even if resource fails.

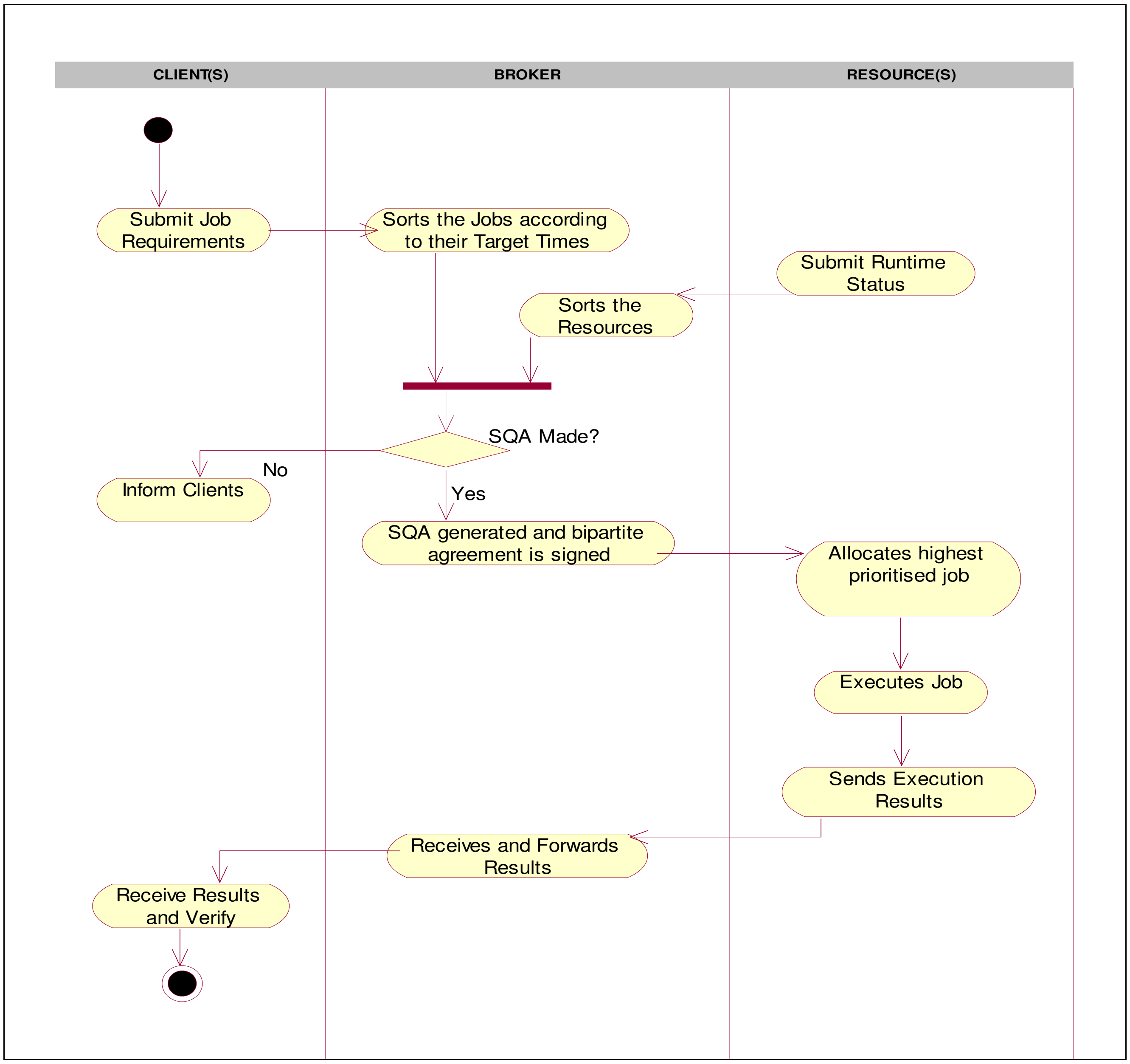
CLIENT(S)
BROKER
RESOURCE(S)
Submit Job Requirements
Sorts the Jobs according to their Target Times
Submit Runtime Status
Sorts the Resources
SQA Made?
No
Inform Clients
Yes
SQA generated and bipartite agreement is signed
Allocates highest prioritised job
Executes Job
Sends Execution Results
Receives and Forwards Results
Receive Results and Verify

Figure 1 Activity Diagram of NDFS

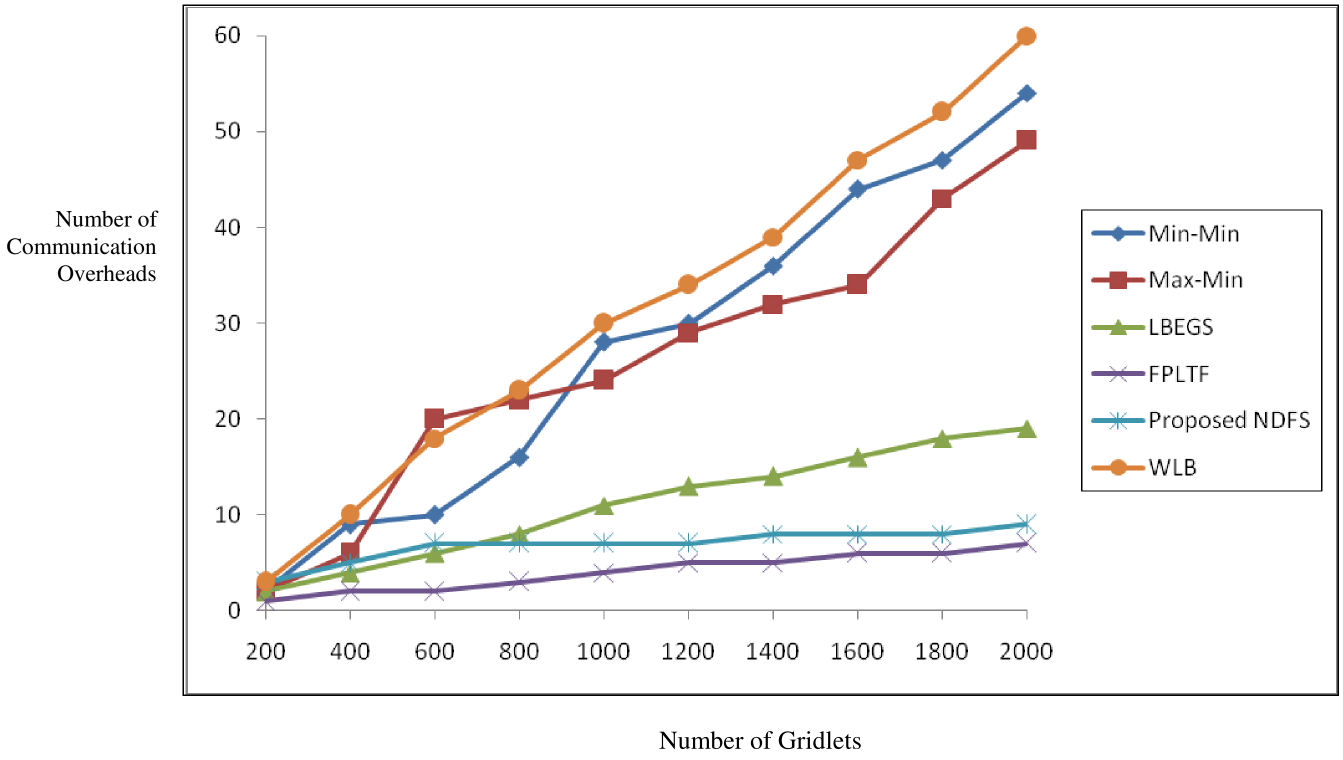

Figure 2 Comparison of Communication Overhead Times versus total number of Gridlets, PEs=200

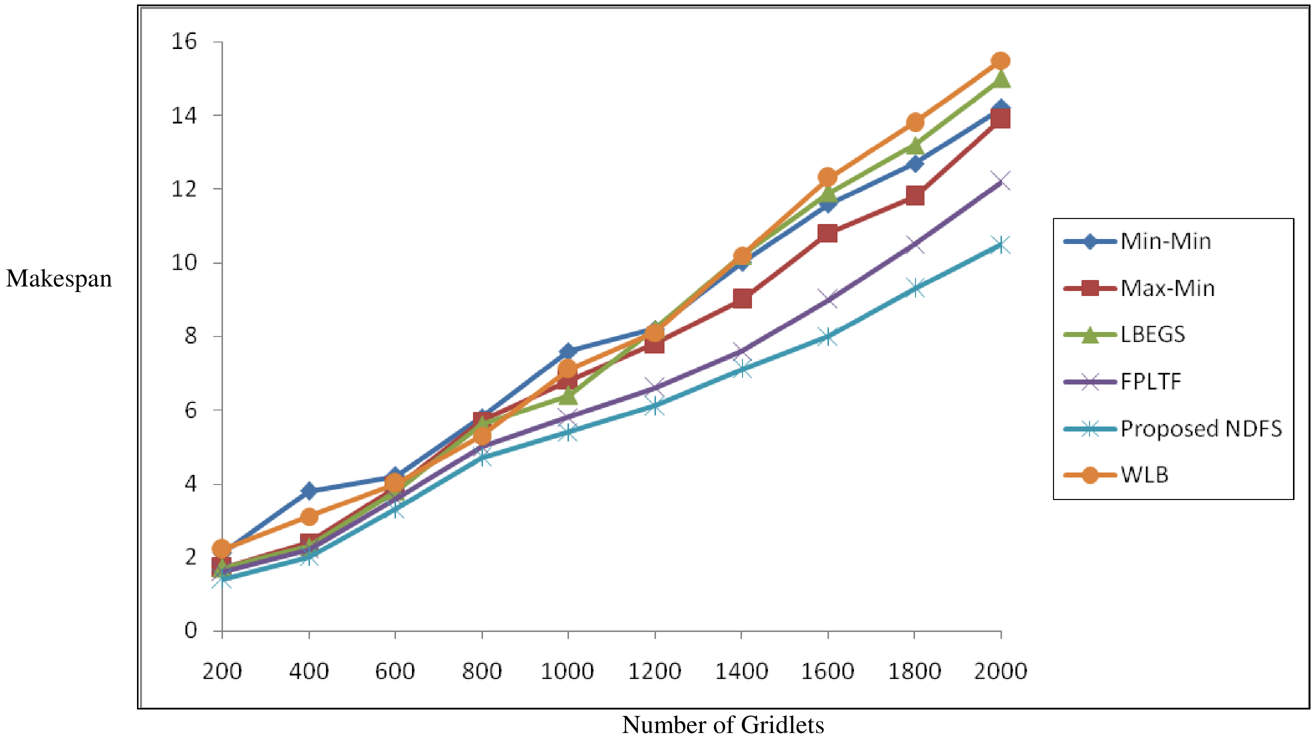

Figure 3 Comparison of Makespan versus total number of Gridlets, PEs=200

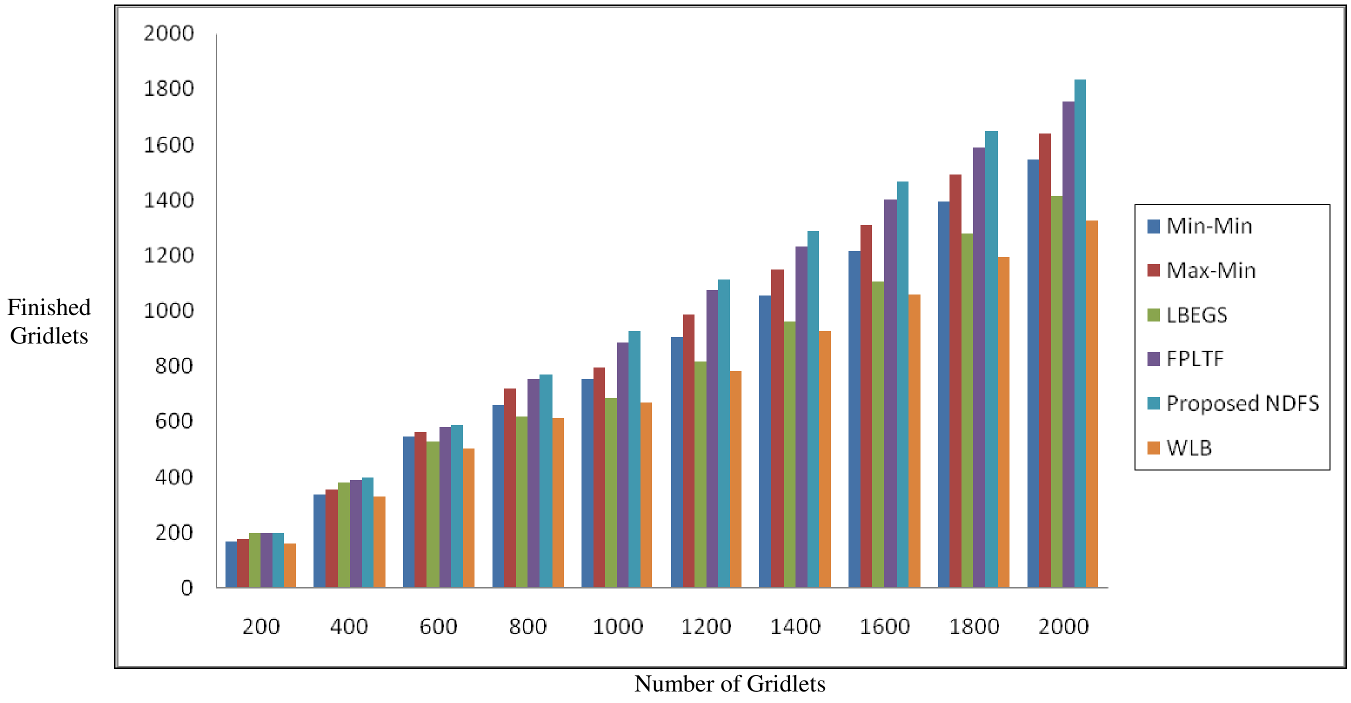

Figure 4 Comparison of No. of finished Gridlets versus total number of Gridlets, PEs=200

REFERENCES

[1] I. Foster, C. Kesselman, S. Tuccke, "The Anatomy of the Grid", International Journal of Supercomputer Applications, 2001.

[2] J. Cao, Self-organizing agents for grid load balancing, in: Proceedings of the fifth IEEE/ACM International Workshop on Grid Computing, GRID'04, Pittsburgh, PA, November 2004.

[3] B. Yagoubi, Y. Slimani, Dynamic load balancing strategy for grid computing, Engineering and Technology, 2006.

[4] D. Erdil, M. Lewis, Dynamic grid load sharing with adaptive dissemination protocols, The Journal of Supercomputing, pp. 1–28, 2010.

[5] Kesler, J. C., Overview of Grid Computing, MCNC, 2003.

[6] J. Cao, D. P. Spooner, S. A. Jarvis, G. R. Nudd, Grid load balancing using intelligent agents, Future Generation Computer Systems (ISSN: 0167-739X) 21 (1) 135–149, 2005.

[7] V. Kant Soni, R. Sharma and M. Kumar Mishra, "An analysis of various job scheduling strategies in grid computing", 2nd International Conference on Signal Processing Systems (ICSPS), 2010.

[8] J. Balasangameshwara, N. Raju, A hybrid policy for fault tolerant load balancing in grid computing environments, Journal of Network and Computer Applications 35 pp. 412–422, 2012.

[9] R. Rajavel, De-Centralized Load Balancing for the Computational Grid Environment, International Conference on Communication and Computational Intelligence, Tamil Nadu, India, 2010.

[10] F. Dong and S. G. Akl, Scheduling Algorithms for Grid Computing: State of the Art and Open Problems, Technical Report No. 2006-504, School of Computing, Queen's University, Kingston, Ontario, January 2006.

[11] S. Zikos, H.D. Karatza, Communication cost effective scheduling policies of non-clairvoyant jobs with load balancing in a Grid, Journal of Systems and Software 82, 2009, 2103–2116.

[12] M. Mezmaz, N. Melab, E.G. Talbi, An efficient load balancing strategy for Gridbased branch and bound algorithm, Parallel Computing 33, 2007, 302–313.

[13] J. Balasangameshwara, N. Raju, Performance-Driven Load Balancing with Primary-Backup Approach for Computational Grids with Low Communication Cost and Replication Cost, IEEE Transactions on Computers, Digital Object Identifier 10.1109/TC.2012.44, 2012.

[14] S. Goswami, A. De Sarkar, Service Oriented Load Balancing Framework in Computational Grid Environment, International Journal of Computers and Technology, Volume 9, Number 3, pp 1091 – 1098, 2013.

[15] S. Goswami, A. De Sarkar, A Comparative Study of Load Balancing Algorithms in Computational Grid Environment, Fifth International Conference on Computational Intelligence, Modeling and Simulation, CIMSIM-2013.

[16] R. Buyya, M. Murshed, GridSim: a toolkit for the modeling and simulation of distributed management and scheduling for Grid computing, The Journal of Concurrency and Computation: Practice and Experience 14 pp. 13–15, 2002.

[17] K. Qureshi, A. Rehman, P. Manuel, Enhanced GridSim architecture with load balancing, The Journal of Supercomputing pp. 1–11, 2010.

[18] www.java.com

[19] www.eclipse.org