Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 2, Issue. 8, August 2013, pg.7 – 13

RESEARCH ARTICLE

Backfilling Strategies for Computational Grid System Load Balancing

Prakash Kumar[1], Pradeep Kumar[2], Vikas Kumar[3]
[1]CSE Department, MTU, India
[2]CSE Department, MTU, India
[3]Eurus Internetworks, India

[1] Prakashkumar.ce@gmail.com; [2] pradeep2345in@gmail.com; [3] vikas.kumarsec@gmail.com

Abstract— Grid is distributed computing infrastructure for advanced science and engineering that runs over the internet, potentially world-wide. Grid is highly controlled, with resource providers and consumers defining what is shared and the conditions of sharing. The goal of Grid computing is to create the delusion 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. Backfill is a scheduling optimization which allows a scheduler to make better use of available resources by running jobs out of order. Backfill locates jobs to run from throughout the idle job queue, it tends to moderate the influence of the job prioritization a site has chosen and thus may negate some desired workload steering attempts through this prioritization. Essentially filling in holes in node space, backfill tends to favor smaller and shorter running jobs more than larger and longer running ones.

Key Terms: - Grid computation; Load balancing; Scheduling; Cluster; Throughput; Conservative Backfilling algorithm; GAP Search

I. INTRODUCTION

Concepts of Grid Computing explores an emerging technology that enables large-scale resource sharing problem solving within distributed, loosely coordinated groups sometimes termed virtual organizations. Grid Computing is a cost efficient solution with respect to super Computing. Grid distinguish Grid computing from conventional high performance computing systems such as cluster computing is that grids tend to be more loosely coupled, heterogeneous, and geographically dispersed. A grid job that enters the system will first be dispatched to a specific site by the grid scheduler, and then be further allocated to a processor by the local scheduler [1]. Research into scheduling for the grid environment can be broadly classified into two categories i.e. Application-level scheduling, the focus is on approaches to optimize the performance of a single job in a grid environment, and Job-level scheduling, the focus is on the performance optimization across a collection of independent jobs[4]. The notion of Grid computing extends well beyond the traditional Parallel and Distributed Computing Systems, as it involves various resources that belong to different administrative domains and are controlled by domain specific resource management policies [5]. In order to utilize these dynamic resources and jobs optimally, a scheduling strategy should be able to continually adapt to the changes and properly distribute the workload and data amounts scheduled to each node [11]. The end goal of the work is to provide an integrated Grid application development solution that incorporates activities such as compilation, scheduling, staging of binaries and data, application launch, and monitoring of application progress during execution [15]. Enabling backfill allows the scheduler to start other, lower-priority jobs so long as they do not delay the highest priority job. Enabling backfill increases system utilization by about 25% and improves turnaround time by an even greater amount. GridSim is the shape of heterogeneous, multitasking Grid resources, calendar based on deadline and budget based constraints. It also allows evaluating various scheduling policies in a single simulation. Each Grid task is implemented as a separate thread in the Java Virtual Machine [6].

II. PREVIOUS WORK

The First Come First Served (FCFS) algorithm has proven insufficient and can cause severe fragmentation when resources are not available for large gangs [1]. Considering Random search, the next job to be scheduled is randomly selected among all jobs that are submitted but not yet started, therefore the schedule is non-deterministic. No job is preferred, but jobs submitted earlier have a higher probability to be started before a given time instant [2]. The greedy multisite scheduling algorithm scales well while the optimal one does not have polynomial time complexity. Initial experimental results show that the adaptability of an algorithm is very important to its performance, as shown by comparing optimal and greedy adaptive algorithms with the non-adaptive version [4]. Proposed a distributed load balancing model which takes into account the heterogeneity of the resources that is completely independent from any physical architecture Grid [6]. Co-allocation algorithm to reduce the total time to release user jobs and waiting time in the global queue, maximize the resources utilization rate and load the balance among the resources providers, and compared our results with FCFS, EDF algorithms [7]. Different simulation experiments are performed to compare different aspects of scheduling using different types of job, resources and workloads. The execution results indicate the effect of scheduling approach used on efficient execution of grid jobs, success ratio of jobs with QoS requirement and resource utilization and load balancing of grid system [9].

III. SCHEDULING ARCHITECTURE

The scheduler is made of two main components: the Scheduler and the Resource Manager. Each of them has its own functionality; the scheduler is in charge of registering jobs submitted and put them in a queue according to a scheduling policy. Then, it has to ask for resources at the Resource Manager, and execute jobs on those retrieved resources; the Resource Manager (RM) handles a set of available resource available for scheduling jobs. It benefits from the Proactive Library, so it can handle resources from LAN, on cluster of workstations, on P2P desktop Grids, or on Internet Grids. Resource Manager provides the scheduler with resources, according to criteria (Operating System, dynamic libraries, Memory). Resources, at Proactive point of view, are called nodes. Resource Manager therefore supplies Proactive nodes to the Scheduler. We distinguish centralized and decentralized scheduling architectures (Shown in Figure 1).

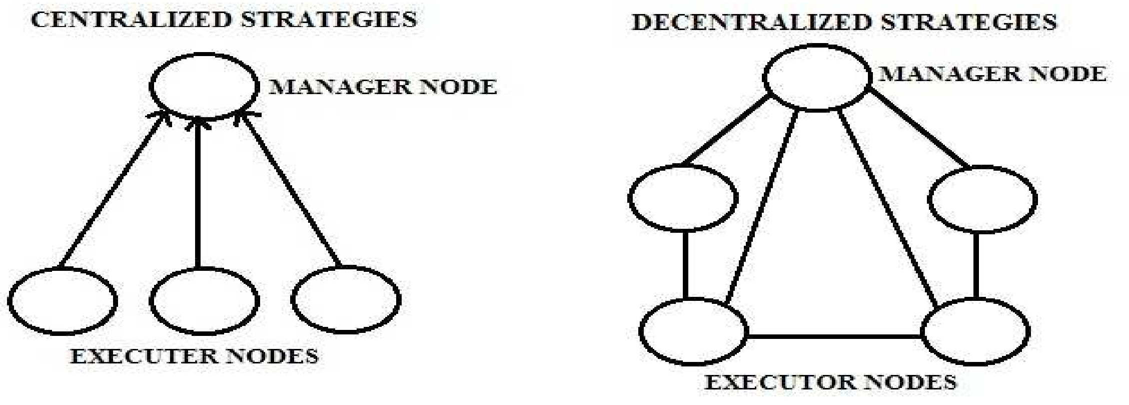

Figure 1: Centralized and De-Centralized Strategies

In a centralized environment all parallel machines are scheduled by a central instance. Information on the state of all available systems must be collected here. This concept obviously does not scale well with increasing size of the computational grid. The central scheduler may prove to be a bottleneck in some situations (e.g. if a network error cuts of the scheduler from its resources, system availability and performance may be affected). As an advantage, the scheduler is conceptually able to produce very efficient schedules, because the central instance has all necessary information on the available resources. The proposed algorithm consists of two parts. Most of the local jobs are scheduled on the local machine with a list-scheduling algorithm working in batches. Moreover, the scheduler attempts to migrate jobs which would miss their due dates when executed locally. An algorithm for handling such migration requests from the receiver's point of view. Although migration improves the performance of the originator, migrated jobs can possibly delay local jobs and, consequently, worsen the local criterion [16].

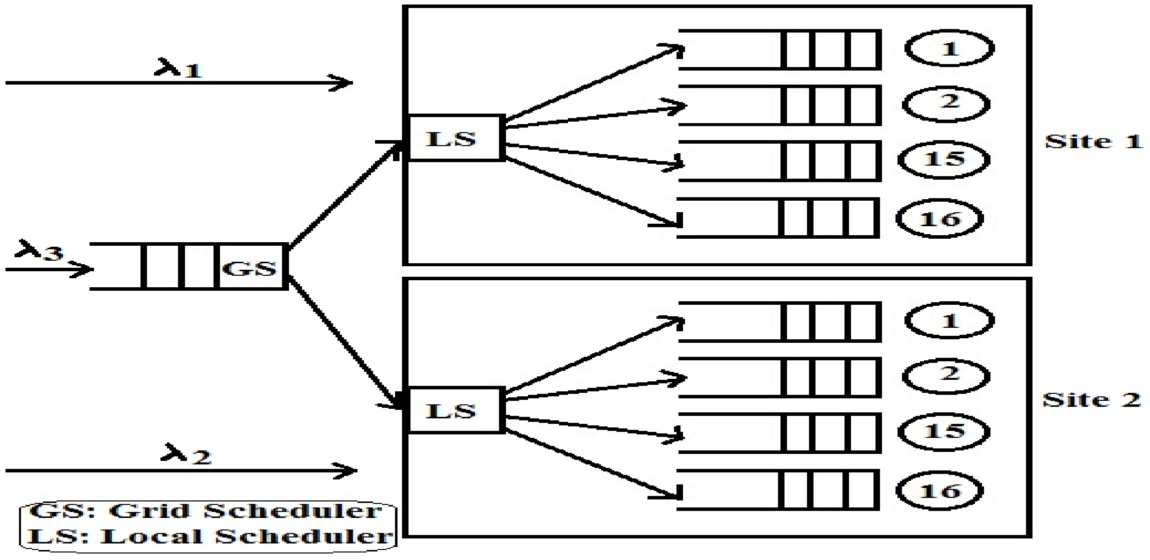

Figure 2: The Queuing Model

The workload consists of two different job types competing for the same resources: local jobs and grid jobs. Therefore, there are three arrival streams in the system: one at the GS (grid jobs) and one inside each of the two sites (local jobs). A local job consists of a single task, while a grid job (gang) consists of a number of parallel tasks that must be processed simultaneously (Shown in Figure 2). The mean inter-arrival time of gangs and locals is exponentially distributed with a mean of $1/\lambda1$ for the locals in site1, $1/\lambda2$ for the locals in site2 and $1/\lambda3$ for the gangs, where $\lambda1$, $\lambda2$ and $\lambda3$ are the arrival rates for locals in site1, site2 and gangs, respectively. We assume that the arrival rates in both sites are the same ($\lambda1=\lambda2=\lambda$), and that the arrival rate of grid jobs is much lower than that of local jobs ($\lambda3 << \lambda$) [1]. Super scheduling in computational grids is facilitated by specialized super schedulers such as Grid Federation Agent [5].

IV. Load Balancing Strategies

In accordance with the structure of the proposed model, we distinguish between two load balancing levels: Intra-cluster (Inter-worker nodes) and Inter-clusters (Intra-Grid) load balancing. We suppose that computing elements have different characteristics (speed, period for sending load information); Tasks are independent; it is natural to have different communication costs between clusters, because of the heterogeneity of WANs. In intra-cluster communication costs are the same, since each cluster has a network (LAN) providing similar bandwidths for all its worker nodes. In contrast to the intra-cluster level, we should consider the communication cost among clusters. Knowing the global state of each cluster, the overloaded cluster manager can distribute its overloaded tasks between under-loaded clusters [6, 8].

First Come First Serve (FCFS):

First Come First Serve (FCFS) or also known as First in First out (FIFO) is the simplest and the most fundamental of grid scheduling that involves client-server interaction. In grid scheduling, FCFS policy manages the jobs based on their arrival time, which means that the first job will be processed first without other biases or preferences.

Shortest Job First (SJF)

Shortest job First (SJF) also known as Shortest Job Next (SJN) or Shortest Process Next (SPN) is a scheduling technique that selects the job with the smallest execution time. The jobs are queued with the smallest execution time placed first and the job with the longest execution time placed last and given the lowest priority [11].

Earliest Gap-Earliest Deadline First Rule (EG-EDF):

It places a new submitted job into the existing schedule to build the schedule incrementally. It permits us to compute a new job scheduling plan saving running time for scheduling since the new plan is not re-computed from scratch. To do this, it is necessary to choose a good place in the schedule for the job being scheduled; otherwise resource utilization may drop quickly due to the gaps appearing in the schedule. A gap is considered to be a period of idle CPU time. A new gap appears in the schedule every time the number of currently available

CPUs by the machine is lower than the number of CPUs requested by a job. In such situation job has to be placed in the schedule to a time when a sufficient number of CPUs is available. Gaps can also appear when there are more CPUs than required by the jobs [3].

Backfilling Strategies:

Backfill is a scheduling optimization which allows a scheduler to make better use of available resources by running jobs out of order [14]. Schedulers employing backfilling algorithms in Distributed-Memory Parallel System have been found to improve system utilization and job response time by allowing smaller jobs from back of the waiting queue to execute before the larger jobs that have arrived earlier. By arranging jobs in a specific order, we reduce internal fragmentation and improve utilization of the system. Backfilling algorithms also overcome the problem of starvation and waste of processing resources exhibited by algorithm like Shortest Job First (SJF). Conservative and aggressive backfilling algorithms usually use a single queue and ignore user priority1. Utilization of the system resources depends on how the jobs are packaged and the order of their execution. We have implemented the backfilling scheduling algorithms using multiple-queue and dynamic algorithms using two look-ahead strategies [12].

Rescheduling Architecture:

Over time, other applications may introduce load in the system or application requirements may change. To sustain good performance for longer running applications, the schedule may need to be modified during application execution.

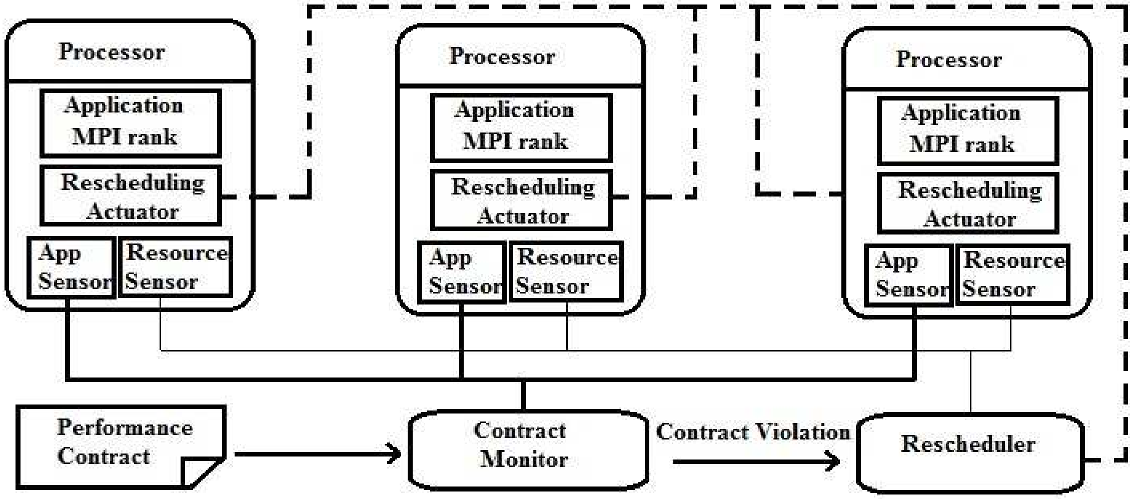

Figure 3: Re-scheduling Architecture

Rescheduling involves a number of complexities not seen in launch-time scheduling. First, while nearly all parallel applications support some form of launch-time scheduling (selection of machines at a minimum), very few applications have built-in mechanisms to support migration or dynamic load balancing. Second, for resource monitoring we have to differentiate between processors on which the application is running and processors on which the application is not running. Measurements from resource monitors such as NWS CPU sensors cannot be directly compared between active and inactive processors. Third, the overheads of rescheduling can be high: monitoring for the need to reschedule is an ongoing process and, when a rescheduling event is initiated, migration of application processes or reallocation of data can be very expensive operations. Without careful design, rescheduling can in fact hurt application performance.

V. PROBLEM STATEMENT

Workload and resource management are two essential functions provided at the service level of the Grid software infrastructure. The focus of our study is to consider factors which can be used as characteristics for decision making to initiate Load Balancing. Load Balancing is one of the most important factors which can affect the performance of the grid application. The main objective of this thesis is to propose an efficient Load Balancing Algorithm for Grid environment. Main difference between existing Load Balancing algorithm and proposed Load Balancing is in implementation of three policies: Information Policy, Triggering Policy and Selection Policy. The paper aims is to design and development of a performance efficient Load Balancing algorithm which overcomes the shortcomings of the current state of the art in the context.

VI. PROPOSED METHODOLOGY

The choice of a load balancing algorithm for a Grid environment is not always an easy task. Various algorithms have been proposed in the literature, and each of them varies based on some specific application domain. Some strategies are focused towards handling data-heavy tasks, while others are more suited to parallel tasks that are computation heavy. While many different load balancing algorithms have been proposed, there are four basic steps that nearly all algorithms have in common: Monitoring workstation performance (load monitoring); Exchanging this information between workstations (synchronization); Calculating new distributions and making the work movement decision(rebalancing criteria); Actual data movement (job migration). The approach aim is, all the jobs should be executed which are submitted to central schedulers by sending the jobs which cannot be executed immediately to the job pool and by implementing special policies on all those jobs which cannot be executed immediately [13]. Within the context of scheduling resources in a computational grid, we supplement the single and multiple-queue backfilling policies by considering two static job priority levels. We consider those jobs submitted by local users to have high priority and those jobs submitted externally (i.e., from elsewhere in the computational grid) to have low priority [10]. The GridSim toolkit provides a comprehensive facility for simulation of different classes of heterogeneous resources, users, applications, resource brokers, and schedulers. It can be used to simulate application schedulers for single or multiple administrative domains distributed computing systems such as clusters and Grids.

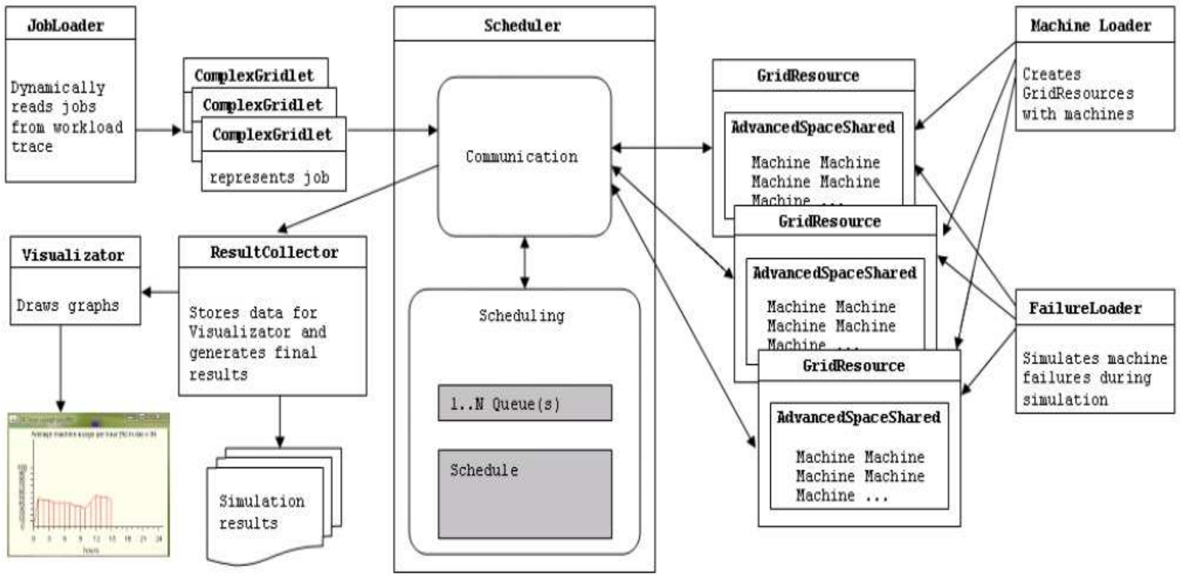

Figure 4: The process of simulation using GridSim Simulator

Backfilling improves resource utilization by allowing small job to fill in those gaps. Job which is lower in the queue is moved to the idle machines without delaying the execution of the job at the top of the queue. The algorithms works by rearrange existing jobs queue based on increasing order of the execution time of the jobs. Candidate job to be moved forward is the one that is also in the queue [11].

VII. EXPERIMENTAL RESULTS

To implement the proposed Load Balancing Algorithm, an application has been developed, which is executed in simulated grid environment. Java language offers several features that facilitate with easiness the development and deployment of a software environment for Grid computing. As Grid is network based java's network based features are very useful to develop Grid application. Java's network-centric approach and its built-in support for mobile code enable the distribution of computational tasks to different computer platforms.

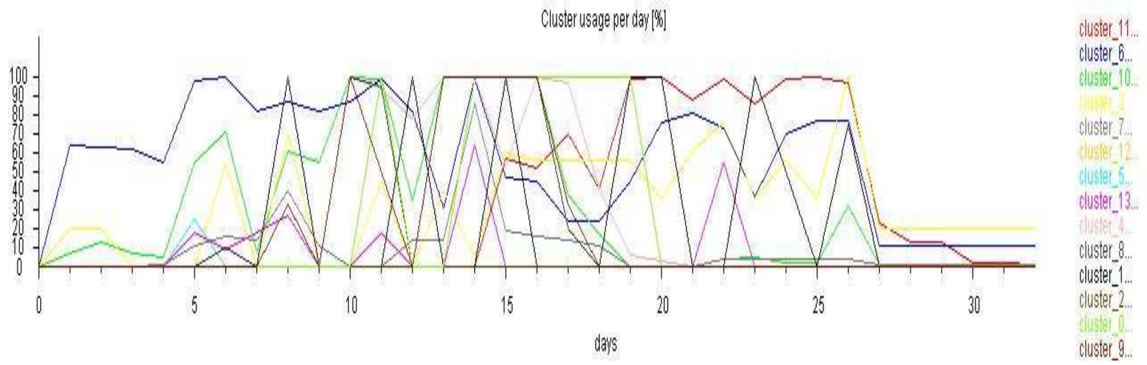

Figure 5: Percentage of Cluster usage per day in previous algorithm

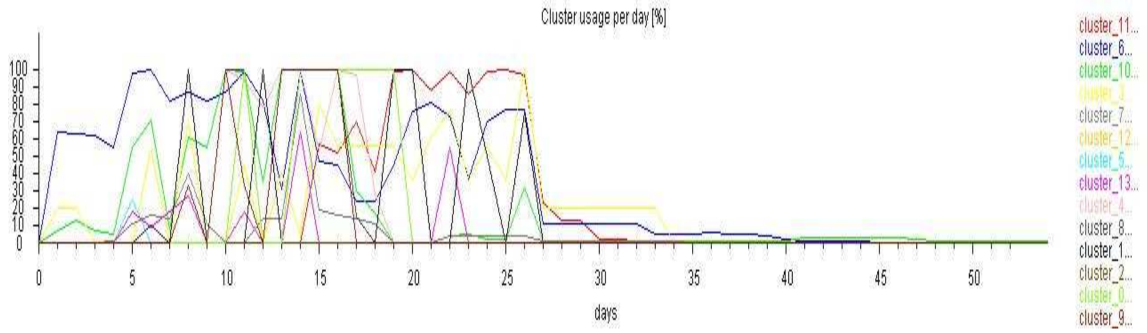

Figure 6: Percentage of Cluster usage per day in proposed algorithm

The GridSim toolkit provides a comprehensive facility for simulation of different classes of heterogeneous resources, users, applications, resource brokers, and schedulers. It can be used to simulate application schedulers for single or multiple administrative domains distributed computing systems such as clusters and Grids.

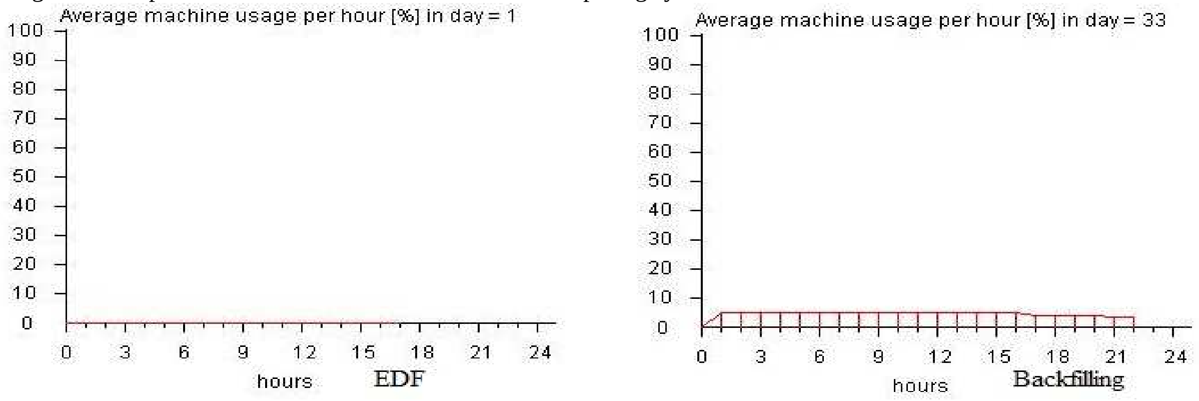

Figure 7: Percentage of Average machine usage per hour

The experimental result shows the comparison between previous algorithm and proposed algorithm with respect to percentage of cluster uses per day and number of waiting/running jobs per day. Figure 5 and 6 shows the result of proposed algorithm is better than the past. Figure 7 shows the improvement of the percentage of average machine usage per hour.

VIII. Conclusions and Future Scope

Grid Computing is definitely a promising tendency to solve high demanding applications and all kinds of problems. Objective of the grid environment is to achieve high performance computing by optimal usage of geographically distributed and heterogeneous resources. We analyzed existing Load Balancing algorithm and proposed an enhanced algorithm which more efficiently implements three out of five policies implemented in existing Load Balancing algorithm. In future, we will improve the performance of grid application but also makes it more powerful, reliable and capable of handling more complex and large problems in Grid environment.

REFERENCES

[1] Sofia K. Dimitriadou, Helen D. Karatza, "Job Scheduling in a Distributed System Using Backfilling with Inaccurate Runtime Computations", International Conference on Complex, Intelligent and Software Intensive Systems, IEEE, 2010.

[2] Volker Hamscher, Uwe Schwiegelshohn, Achim Streit, Ramin Yahyapour, "Evaluation of Job-Scheduling Strategies for Grid Computing", ACM International Workshop on Grid Computing, 2000.

[3] Ranieri Baraglia, Marco Pasquali, Gabriele Capannini, "Comparison of Multi-Criteria Scheduling Techniques", Institute of Italian National Research Council, 2008.

[4] Weizhe Zhang, Albert M.K Cheng, Mingzeng Hu, "Multisite Co-allocation Algorithms for Computational Grid", IEEE, 2006.

[5] Rajiv Ranjan, Aaron Harwood, Rajkumar Buyya, "SLA-Based Cooperative Super scheduling Algorithms for Computational Grids", ACM, Volume 5, Issue N, August 2006.

[6] Belabbus Yagoubi, Meriem Meddeber, "Distributed Load Balancing Model for Grid Computing", ARIMA Journal, Vol. 12, Pages 43-60, September 2010.

[7] Sid Ahmed Makhlouf, Belabbas Yagoubi, "Co-allocation in Grid Computing using Resources Offers and Advance Reservation Planning", Courrier du Savoir, November 2012.

[8] Ye Huang, Nik Bessis, Peter Norrington, Pierre Kuonen, Beat Hirsbrunner, "Exploring decentralized dynamic scheduling for grid and clouds using the community-aware scheduling algorithm", Elsevier, November 2010.

[9] Aditya B patel, "Modeling and Simulation of Grid Resource Brokering Algorithms", International Journal of Computer Application, Volume 42, Issue 8, March 2012.

[10] Barry G Lawson, Evgenia Smirni, "Multiple-queue Backfilling Scheduling with Priorities and reservations for Parallel Systems", Springer, Volume 2537, 2002.

[11] Zafril Rizal M Azmi, Kamalrulnizam Abu Baker, Mohd Shahir Shamsir, Wan Nurulsafawati Wan Manan, Abdul Hanan, Abdullah, "Scheduling Grid Jobs Using priority Rule Algorithms and Gap Filling Techniques", International Journal of Advanced Science and Technology, Volume 37, December, 2011.

[12] Hasasn rajaei, Mohammad Dadfar, "Comparison of Backfilling Algorithms for Job Scheduling in Distributed Memory Parallel System", American society for Engineering Education, 2006.

[13] Edi Laxmi, "International Journal of Computer Science and Management Research", IJCSMR, Volume 2, Issue 1, January 2013.

[14] David Jackson, Quinun Snell, Mark Clement, "Core Algorithms of the Maui Scheduler", JSSPP, Springer, Volume 2221, 2001.

[15] Holly Dail, Fran Berman, Henri Casanova, "A decoupled scheduling approach for Grid application development environments", Elsevier, April 2002.

[16] Marchin Krystek, Krzysztof Kurowski, Ariel Oleksiak, "Comparison of Centralized and De-centralized Scheduling Algorithms using GSSIM Simulation Environment", Springer, 2008.