Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2013, Article ID 439362, 8 pages
http://dx.doi.org/10.1155/2013/439362

Research Article

User Demand Aware Grid Scheduling Model with Hierarchical Load Balancing

P. Suresh[1] and P. Balasubramanie[2]

[1] *Department of Information Technology, Kongu Engineering College, Perundurai, Erode, Tamil Nadu 638052, India*
[2] *Department of Computer Science and Engineering, Kongu Engineering College, Perundurai, Erode, Tamil Nadu 638052, India*

Correspondence should be addressed to P. Suresh; sureshme@gmail.com

Received 15 March 2013; Accepted 19 May 2013

Academic Editor: Tingwen Huang

Grid computing is a collection of computational and data resources, providing the means to support both computational intensive applications and data intensive applications. In order to improve the overall performance and efficient utilization of the resources, an efficient load balanced scheduling algorithm has to be implemented. The scheduling approach also needs to consider user demand to improve user satisfaction. This paper proposes a dynamic hierarchical load balancing approach which considers load of each resource and performs load balancing. It minimizes the response time of the jobs and improves the utilization of the resources in grid environment. By considering the user demand of the jobs, the scheduling algorithm also improves the user satisfaction. The experimental results show the improvement of the proposed load balancing method.

1. Introduction

Grid computing is a new computing paradigm which provides massive heterogeneous resources for solving very large applications in science and engineering [1]. It mainly deals with large scale computing and scientific problems. Grid is divided into two categories such as computational grid and data grid. A computational grid is a collection of hardware and software resources that provide an inexpensive access to all high end capabilities [2]. It supports sharing and coordinated use of computational resources which are distributed geographically [3]. Data grid focuses on controlling and accessing of massive source of data storage.

Task scheduling is a challenging task in grid environment because it is needed to exploit the capabilities of the grid resources. An efficient scheduling algorithm has to allocate the resources from various administrative domains to large number of tasks [4]. The load of resource needs to be considered to increase the performance, resource utilization, and throughput. Task scheduling involves mapping of "n" tasks to "m" resources. It is a NP-complete problem. Scheduling is done using an application called scheduler. The scheduler software enables an enterprise to schedule and in some cases monitor computer "batch" tasks [5].

Scheduling algorithms are categorized as application centric and resource centric. Scheduling algorithms adopting resource centric objectives aim to improve the performance of the system such as resource utilization and throughput. Scheduling algorithms adopting application centric objectives aim to minimize the waiting time and makespan.

Generally, load balancing algorithms are divided into two categories such as static and dynamic. Static load balancing algorithms are used to allocate the resources to tasks based on the current load. Dynamic load balancing algorithms allocate or reallocate resources at runtime based on the current load of the resources [6]. Load balancing algorithms aim to maximize the number of jobs completion, resource utilization, and minimizing the makespan. This section describes many task scheduling algorithms which are used to allocate resources to jobs.

Opportunistic load balancing (OLB) algorithm allocated the task to the next machine which is expected available in arbitrary order. This algorithm did not consider the expected execution time on the machine [7]. Minimum execution time

(MET) algorithm scheduled tasks by considering expected execution time regardless of availability of machine. It causes poor makespan and severe load imbalance [8]. Minimum completion time (MCT) algorithm allocated job in random order to the resource which has minimum completion time. But it leads to load imbalance [9].

Min-min algorithm calculated expected completion time for all the jobs in all machines. Task with minimum expected completion time is allocated to a machine which has minimum completion time. This algorithm failed to consider idleness of the machine [9, 10].

Max-min algorithm is similar to min-min algorithm. It finds the job with maximum expected completion time from set of unmapped jobs and allocates it to machine which has minimum completion time for the job [11].

Application demand aware algorithm considered the application's requirement and allocated the jobs to the appropriate resources. But results show poor resource utilization [12].

Suffrage heuristic algorithm calculated the suffrage value by finding the difference between its best minimum completion time and its second best minimum completion time. The task with highest suffrage value is selected and allocated to the machine with the minimum completion time. The assigned task is deleted, and the completion times for all the remaining tasks are updated. This process is repeated until all tasks are assigned to machines [13].

Prioritized user demand aware algorithm allocated jobs to the machines by considering user requirements of jobs. It results in more user satisfaction and better makespan, but data movement is not considered [14].

UDDA algorithm considered both user requirements and data requirements of the job. It aims to reduce the overall execution time of the jobs in the machines [15].

Yagoubi and Slimani developed a layered algorithm in which a dynamic load balancing approach is implemented in grid computing. Load balancing is performed based on a tree model. It is supported for heterogeneity and scalability [16].

Ludwig and Maollem proposed two new distributed swarm intelligence inspired load balancing algorithms. One algorithm is based on ant colony optimization, and the other algorithm is based on particle swarm optimization [17].

Cao et al. designed a load balancing algorithm in combination with intelligent agents and multiagent approaches. This algorithm assumed two levels such as local grid and global grid. In local grid, each agent is responsible for scheduling and load balancing, and in global grid, agent acts as service provider [18].

Alharbi et al. proposed a simple load balancing algorithm in which load is equally distributed. This algorithm used expected completion time to schedule the jobs [19].

Even though there are many scheduling algorithms that focus on user deadline and load balancing, there is a scope for scheduling algorithms that focuses on both of these factors. The main contribution of this paper is that a new scheduling algorithm has been proposed which focuses on both user deadline consideration and load balancing. This algorithm is compared with the min-min algorithm which serves as a benchmark scheduling algorithm and user demand aware (UDDA) scheduling algorithm which is proposed in our previous work [15]. The UDDA algorithm considers only the user deadline and do not consider the load balancing factor. The proposed algorithm considers both user satisfaction and load balancing.

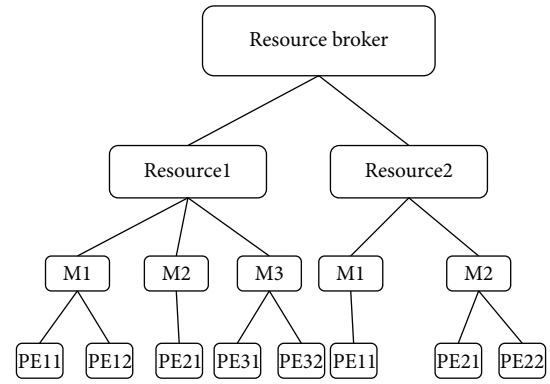

Figure 1: Proposed Architecture Model.

2. Materials and Methods

2.1. Problem Definition. Since grid environment consists of heterogeneous resources which are owned by multiple administrative domains, an efficient task scheduling is required for allocating tasks to machines efficiently. Since grid resources are dynamic, an effective load balancing algorithm is required to maintain load at resources. Most of the early developed scheduling algorithms do not consider the user deadline to complete the jobs. To improve the user satisfaction, the deadline of the jobs has to be considered.

The architecture followed in this work is given in Figure 1. Grid resource broker collects information about the resources such as capacity expressed in million instructions per second (MIPS), cost, and baud rate. Grid resource is an entity which is next to the grid resource broker in the hierarchy. It is responsible for maintaining load at the machines which are next to the resource in the hierarchy. Machines are responsible for scheduling and maintaining balanced load at the processing elements.

Users submit jobs in the form of gridlets to the machines. A gridlet is an entity which contains information such as length expressed in MI (million instructions), input file size (IF_i) expressed in MB, output file size (OF_i) expressed in MB, and the user who submits the job. This information is used to calculate expected execution time, data transfer time between the user and the resource which are considered to select the appropriate resources for the jobs.

The length of the gridlet is assumed to be calculated based on the number of instructions. The scheduling algorithm is applied to a batch of jobs submitted since it follows batch mode of scheduling, and the user deadline based selection policy is followed when jobs are waiting in queue at the scheduler.

2.2. Hierarchical Load Balancing Algorithm (HLBA). The proposed scheduling algorithm mainly aims for effective

Begin
  While there are jobs in unassigned list
  do
    Select job i which has minimum user deadline from unassigned list
    For all resources j in the resource list
    do
      $\mathrm{EET}(i, j) = \dfrac{\mathrm{Length}_i}{\mathrm{Capacity}_j}$
      $S_i = \mathrm{IF}_i + \mathrm{OF}_i$
      $\mathrm{DT}(i, j) = \dfrac{S_i}{B_j}$ where B_j is the baud rate of resource j expressed in Mbps
      $\mathrm{RT}(j) = \sum_{i=1}^{n} \mathrm{EET}(i, j)$ where n is number of jobs that are already in queue
      $\mathrm{ECT}(i, j) = \mathrm{RT}(j) + \mathrm{EET}(i, j) + \mathrm{DT}(i, j)$
    End for
  Calculate difference matrix such that
    $\mathrm{DF}(i, j) = D_i - \mathrm{ECT}(i, j)$ for all i, j and if $\mathrm{DF}(i, j) < 0$, then assign a maximum value to $\mathrm{DF}(i, j)$
    where i represents the jobs and j is resource/machine/PE
  Classify the resources/machines/PEs using Algorithms 2, 3, 4
  Sort the list of resources/machines/PEs in ascending order based on $\mathrm{DF}(i, j)$
  Assign the job i to the first resource/machine/PE j in under loaded list with load ≤ 1
  Update the load at each resource/machine/PE
  Remove the job i from unassigned list
  End While
End

Algorithm 1: Scheduling algorithm.

resource utilization and minimized makespan. So it considers both application and system aspects. It considers user deadline, expected completion time, data transfer time, and load of each resource. This algorithm calculates the load at the different levels. Since machine and PEs are arranged very close to the resource, data transfer time to machine/PE is negligible, the resources are geographically distributed, and data transfer time to the resource is considered.

Steps

(i) Users submit the jobs to the machines and add the jobs to unassigned list.

(ii) Assign the jobs among the processing elements (PEs) of the machine with satisfied user demand using Algorithm 1, and remove the job from unassigned list. If it is not possible, then forward the set of unassigned jobs to the resource.

(iii) Assign the jobs among the machines of the resource with satisfied user demand using Algorithm 1, and remove the job from unassigned list. If it is not possible, then forward the set of unassigned jobs to the resource broker.

(iv) Assign the jobs among the resources with satisfied user demand using Algorithm 1, and remove the job from unassigned list.

In this model, number of machines, number of tasks, and size of tasks are known prior. The following parameters are used in this algorithm.

$\mathrm{EET}(i, j)$: expected execution time of task i in resource j.

$\mathrm{ECT}(i, j)$: expected completion time of task i in resource/machine/PE_j.

$\mathrm{RT}(j)$: ready time of the resource/machine/PE_j.

D_i: user demand or deadline of the task i.

$\mathrm{DT}(i, j)$: data transfer time to resource j.

Length_i: length of the job i expressed in MI.

$\mathrm{Capacity}_j$: capacity of the resource/machine/PE_j.

AT_j: availability time of resource/machine/PE_j.

When the user submits jobs at the machines, along with the information such as length, deadline, input, and output file size, machine schedules jobs to their processing elements which are capable of completing the jobs within user deadline. If there is no suitable PE, then the jobs will be forwarded to resources. Then resources will schedule the jobs to machines. If there is no suitable PEs found at all machines under that resource, then resources will forward the jobs to the resource broker which will schedule the jobs to other resources.

Load Balancing. To ensure effective utilization of the resources/machines/PEs, load is calculated at different levels. Load of processing elements (PE) is calculated as follows:

$$\mathrm{PELoad}_j = \frac{\sum_{i=1}^{n} \mathrm{Length}_i}{\mathrm{MIPS}_j \times \mathrm{AT}_j}, \quad (1)$$

```
Begin
    For all resources
        if (RLoad < TRes)
            r.state = "underloaded"
            Add the resource to r.underloaded list
        else if (RLoad > TRes)
            r.state = "heavilyloaded"
            Add the resource to r.heavilyloaded list
        else
            r.state = "normallyloaded"
    End for
End
```

Algorithm 2: State allocation for resource.

where PELoad_j is load of the PE_j, Length_i is length of job i, n is jobs that are submitted to PE_j, and MIPS_j is capacity of PE_j.

Load of the machine is calculated as follows:

$$\text{MLoad}_j = \frac{\sum_{i=1}^{n} \text{Length}_i}{\text{MCapacity}_j \times \text{AT}_j}, \tag{2}$$

where

$$\text{MCapacity}_j = \sum_{i=1}^{m} \text{MIPS}_i, \tag{3}$$

where n is jobs that are submitted to machine_j and m is number of PEs under machine_j.

Load of a resource is calculated as follows:

$$\text{RLoad}_j = \frac{\sum_{i=1}^{n} \text{Length}_i}{\text{RCapacity}_j \times \text{AT}_j}, \tag{4}$$

$$\text{RCapacity}_j = \sum_{i=1}^{m} \text{MCapacity}_i, \tag{5}$$

where n is jobs that are submitted to resource_j and m is number of machines under resource_j.

Since grid is a collection of dynamic resources from different administrative domains, the resources/machines/PEs can be online and offline from the grid. So the status (either online or offline) of the resource/machine/PE must be updated periodically. To perform load balancing, the state of the resource/machine/PE must be checked when a job arrives or status of the resource/machine/PE is changed. The state is classified into three types such as heavily loaded, normally loaded, and under loaded. To assign the state of resource/machine/PE, the threshold value is calculated at different levels. Threshold value for resource is calculated as follows:

$$\text{TRes} = \frac{\sum_{i=1}^{n} \text{RLoad}_i}{n}, \tag{6}$$

where TRes is threshold value for resource and n is number of resources under resource broker. Threshold value for machine is calculated as follows:

$$\text{TMac} = \frac{\sum_{j=1}^{m} \text{MLoad}_j}{m}, \tag{7}$$

where TMac is threshold value for machine and m is total number of machines under a resource. Threshold value for PE is calculated as follows:

$$\text{TPE} = \frac{\sum_{k=1}^{l} \text{PELoad}_k}{l}, \tag{8}$$

where TPE is threshold value for PE and l is number of PEs under a machine.

Based on the threshold value of the load, the state is assigned for resource/machine/PE using the following criteria.

Algorithm 2 is used to categorize the resource based on their states. Algorithm 3 is used to categorize the machines based on their states. Algorithm 4 is used to categorize the processing elements based on their states. When a resource broker/resource/machine gets information about dynamicity (status) of the resource/machine/PE such as newly added or offline, then status is updated using Algorithm 5. Algorithm 6 is used to reschedule the jobs from heavily loaded resource/machine/PE to the resources/machines/PEs in under loaded list.

2.3. Simulation Setup. The proposed HLBA algorithm is simulated using gridsim5.0 toolkit. It is a more popular tool for modeling the resources and scheduling the jobs. It is also used to analyze algorithms on large scale distributed systems of heterogeneous resources. This tool has the facility for creating different classes of heterogeneous resources that can be aggregated using grid resource broker. This tool can be used for solving computational and data intensive applications.

The proposed HLBA algorithm is simulated for varying number of jobs from 100 to 600 in 16 resources. The characteristics of jobs and resources in grid hierarchy are given in Table 1.

```
Begin
    For all resources
        For all machines
            if (MLoad < TMac)
                m.state = "underloaded"
                Add the machine to m.underloaded list
            else if (MLoad > TMac)
                m.state = "heavilyloaded"
                Add the machine to m.heavily loaded list
            else
                m.state = "normallyloaded"
        End for
    End for
End
```

Algorithm 3: State allocation for machine.

```
Begin
    For all resources
        For all machines
            For all PEs
                if (PELoad < TPE)
                    pe.state = "underloaded"
                    Add it to underloaded list
                else if (PELoad > TPE)
                    pe.state = "heavilyloaded"
                    Add it heavilyloaded list.
                else
                    pe.state = "normallyloaded"
            End for
        End for
    End for
End
```

Algorithm 4: State allocation for PE.

Table 1: Characteristics of grid resources and jobs.

Number of resources	16
Number of machines	1–5
Number of PE's per machine	1-2
Number of jobs	100 to 600

The simulation is carried out in two ways. Firstly, the number of resources is kept as 16, the number of jobs is varied from 100 to 600, and the makespan, hit rate, and resource utilization are measured. Secondly, the number of jobs is considered as 512, and the number of resources is considered as 16 which is the benchmark for scheduling algorithms to evaluate their efficiency. The simulation results for these two ways are discussed in Results and Discussion section.

3. Results and Discussion

The performance of the proposed algorithm is evaluated using the parameters such as makespan, hit rate, miss rate, and resource utilization. Makespan is defined as maximum of completion time of the jobs at all resources:

$$\text{Makespan} = \max\left(\text{CT}_i\right), \tag{9}$$

where CT_i is defined as completion time of the last job at resource i.

The utilization of the resources is calculated using the following formulae:

$$\text{RU}_i = \text{load}_i \times 100. \tag{10}$$

Average resource utilization is as follows:

$$\text{ARU} = \frac{\sum_{i=1}^{n} \text{RU}_i}{n}, \tag{11}$$

where RU_i is utilization of the resource i, CT_i is completion time of the last job at resource i, ARU is average resource utilization expressed in percentage, and n is number of resources.

```
Begin
    if (Resource/Machine/PE is newly added)
        Calculate new threshold value at different levels
        Classify the resource/machine/PE based on Algorithms 2, 3, and 4
    Endif
    if (Resource/Machine/PE goes down)
        Retrieves the jobs which are allocated to the resource/Machine/PE and added to unassigned
        list
        Calculate new threshold value at different levels
        Classify the resource/machine/PE based on Algorithms 2, 3, and 4
    Endif
End
```

Algorithm 5: Status update algorithm.

```
Begin
    Select a job from the resource/machine/PE which goes down
    Find the resources/machines/PEs from under loaded list which satisfy user demand for the selected job
    Schedule the jobs among selected resources/machines/PEs using Algorithm 2.
End
```

Algorithm 6: Rescheduling algorithm.

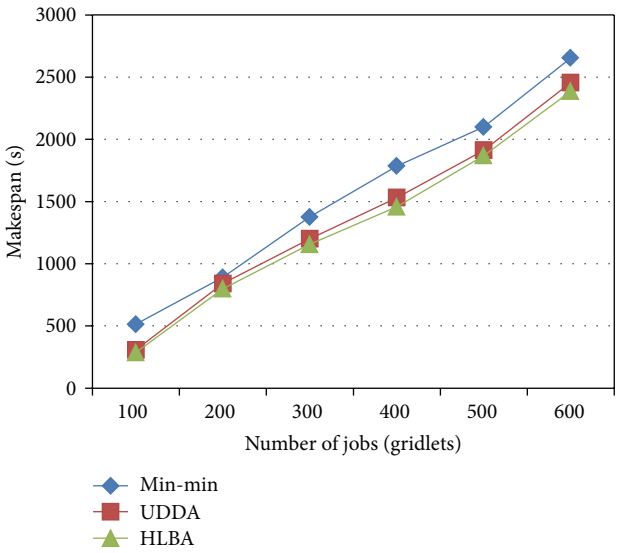

Figure 2: Performance analysis based on makespan (sec).

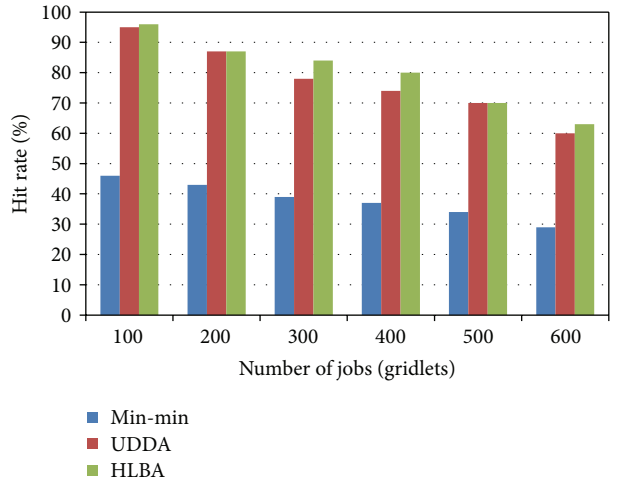

Figure 3: Performance analysis based on hit rate (%).

Hit rate is defined as number of jobs completed within user deadline:

$$\text{Hit rate} = \frac{J_{\text{comp}}}{J_{\text{sub}}} \times 100. \tag{12}$$

Miss rate is defined as number of jobs that could not be completed within user deadline:

$$\text{Miss rate} = \frac{J_{\text{fail}}}{J_{\text{sub}}} \times 100, \tag{13}$$

where J_{comp} is jobs that are completed within user deadline, J_{sub} is jobs that are submitted, and J_{fail} is jobs that could not be completed within user deadline.

The result of the proposed load balancing algorithm is analyzed and compared with min-min and UDDA algorithms. The factors such as resource/machine heterogeneity and job heterogeneity are considered. The improvement of the proposed algorithm is proved by comparing makespan, hit rate, miss rate, and resource utilization. The simulation results have been taken from various test cases. The proposed algorithm generates better results in most of the test cases. Figures 2–5 show the performance of proposed algorithm for varying number of jobs.

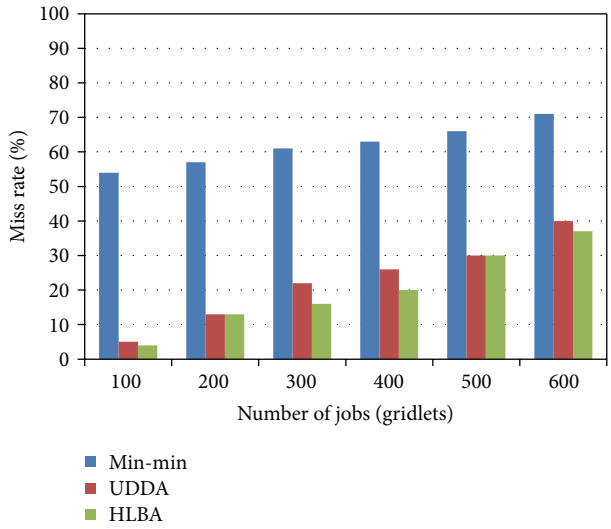

Figure 4: Performance analysis based on miss rate (%).

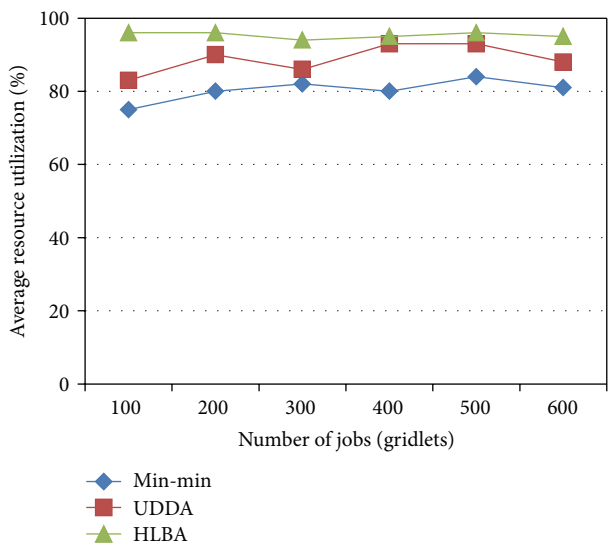

Figure 5: Performance analysis based on average resource utilization (%).

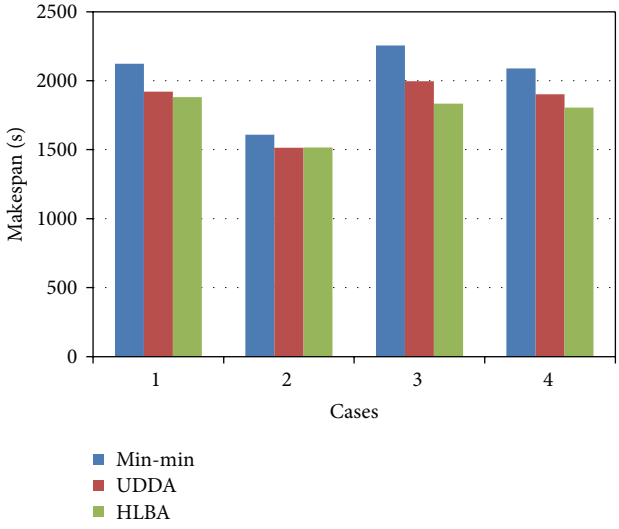

Figure 6: Performance analysis based on makespan (sec).

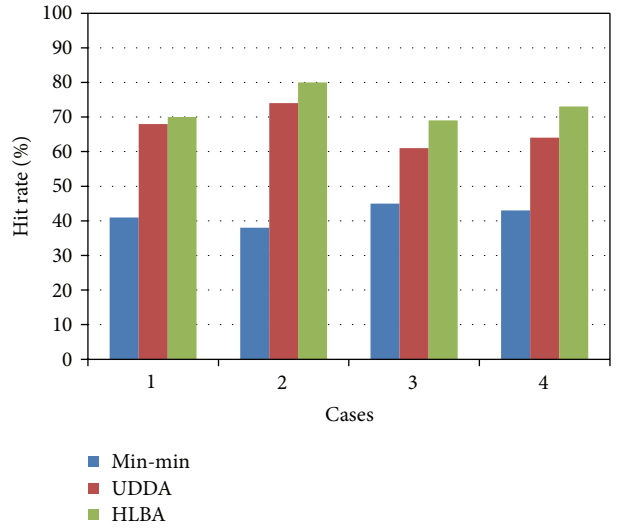

Figure 7: Performance analysis based on hit rate (%).

In Figure 2, the proposed hierarchical load balancing algorithm is compared with the min-min and UDDA algorithm for its efficiency of scheduling which can be measured by makespan expressed in seconds for varying number of jobs from 100 to 600. The performance of the proposed algorithm is better when compared to UDDA and min-min.

In Figure 3, the efficiency of the proposed HLBA algorithm based on hit rate is analyzed, and the analysis shows that the HLBA algorithm has highest hit rate than the other algorithms. Based on miss rate, the performance analysis is shown in Figure 4 which shows that the proposed HLBA has less miss rate than the other algorithms.

The performance analysis of the proposed HLBA and other algorithms min-min and UDDA based on the benchmark values for 512 jobs and 16 resources are elaborated in Figures 6–8. Figure 6 shows that the proposed HLBA has better makespan than min-min and UDDA.

Figures 7 and 8 show that the proposed HLBA has better hit rate and average resource utilization than min-min and UDDA.

4. Conclusion and Future Work

In this paper, hierarchical load balancing approach with user demand aware scheduling algorithm is proposed. This algorithm considers the dynamicity of the resources/machines/PEs, resource/machine heterogeneity, and task heterogeneity. Since this algorithm considers load of resource/machine/PE, the utilization of the resource/machine/PE is improved. By considering the user deadline of the job, the number of jobs completed within user deadline (hit rate) is increased, and user satisfaction is improved. When comparing with other

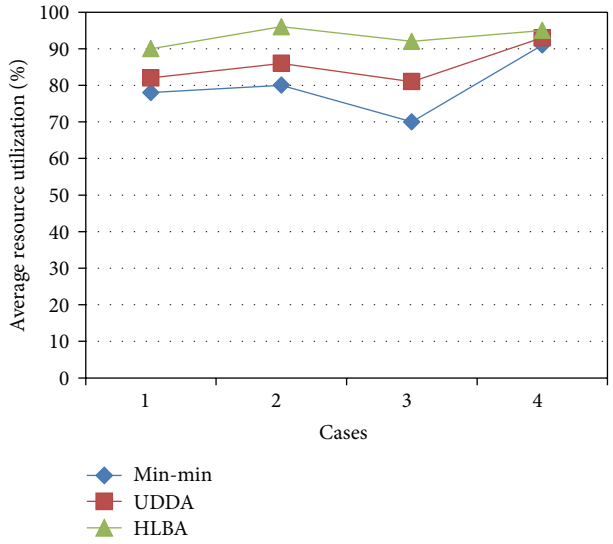

Figure 8: Performance analysis based on average resource utilization (%).

scheduling algorithms, the overall system performance is improved in the proposed scheduling algorithm. In future, some other user demands such as execution cost may be considered, and each machine can be thought as multiprogrammed system. In addition to that some more technologies can also be introduced such as GridFTP and fault tolerant techniques.

References

[1] Q. Zheng, C.-K. Tham, and B. Veeravalli, "Dynamic load balancing and pricing in grid computing with communication delay," *Journal of Grid Computing*, vol. 6, no. 3, pp. 239–253, 2008.

[2] I. Foster and C. Kesselman, *The Grid2: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, Boston, Mass, USA, 2nd edition, 2004.

[3] B. Yagoubi and M. Meddeber, "Distributed load balancing model for grid computing," *ARIMA Journal*, vol. 12, pp. 43–60, 2010.

[4] Y. Li, Y. Yang, M. Ma, and L. 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.

[5] P. Keerthika and N. Kasthuri, "An efficient fault tolerant scheduling approach for computational grid," *American Journal of Applied Sciences*, vol. 9, no. 12, pp. 2046–2051, 2012.

[6] U. Karthick Kumar, "A dynamic load balancing algorithm in computational grid using fair scheduling," *International Journal of Computer Science Issues*, vol. 8, no. 5, article 1, 2011.

[7] T. D. Braun, H. J. Siegel, N. Beck et al., "Comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems," in *Proceedings of the 8th Heterogeneous Computing Workshop (HCW '99)*, pp. 15–29, April 1999.

[8] H. Baghban and A. M. Rahmani, "A heuristic on job scheduling in grid computing environment," in *Proceedings of the 7th International Conference on Grid and Cooperative Computing (GCC '08)*, pp. 141–146, October 2008.

[9] Z. Qian and L. Zhen, "Design of grid resource management system based on divided min-min scheduling algorithm," in *Proceedings of the 1st International Workshop on Education Technology and Computer Science (ETCS '09)*, pp. 613–617, March 2009.

[10] X. He, X. Sun, and G. von Laszewski, "QoS guided Min-Min heuristic for grid task scheduling," *Journal of Computer Science and Technology*, vol. 18, no. 4, pp. 442–451, 2003.

[11] L. Wenzheng and Z. Wenyue, "An improved scheduling algorithm for grid tasks," in *Proceedings of the International Symposium on Intelligent Ubiquitous Computing and Education (IUCE '09)*, pp. 9–12, May 2009.

[12] J. Lin, B. Gong, H. Liu, C. Yang, and Y. Tian, "An application demand aware scheduling algorithm in heterogeneous environment," in *Proceedings of the 11th International Conference on Computer Supported Cooperative Work in Design (CSCWD '07)*, pp. 599–604, IEEE Xplore Press, April 2007.

[13] H. Izakian, A. Abraham, and V. Snášel, "Comparison of heuristics for scheduling independent tasks on heterogeneous distributed environments," in *Proceedings of the International Joint Conference on Computational Sciences and Optimization (CSO '09)*, pp. 8–12, April 2009.

[14] P. Suresh, P. Balasubramanie, and P. Keerthika, "Prioritized user demand approach for scheduling meta tasks on heterogeneous grid environment," *International Journal of Computer Applications*, vol. 23, article 1, 2011.

[15] P. Suresh and P. Balasubramanie, "User demand aware scheduling algorithm for data intensive tasks in grid environment," *European Journal of Scientific Research*, vol. 74, no. 4, pp. 609–616, 2012.

[16] B. Yagoubi and Y. Slimani, "Dynamic load balancing strategy for grid computing," in *Proceedings of the World Academy of Science, Engineering and Technology*, vol. 19, May 2006.

[17] S. A. Ludwig and A. Moallem, "Swarm Intelligence Approaches for Grid Load Balancing," *Journal of Grid Computing*, vol. 9, no. 3, pp. 279–301, 2011.

[18] J. Cao, D. P. Spooner, S. A. Jarvis, and G. R. Nudd, "Grid load balancing using intelligent agents," *Future Generation Computer Systems*, vol. 21, no. 1, pp. 135–149, 2005.

[19] F. Alharbi, "Simple scheduling algorithm with load balancing for grid computing," *Asian Transactions on Computers*, vol. 2, no. 2, 2012.