

A Hierarchical Load Balanced Fault tolerant Grid Scheduling Algorithm with User Satisfaction

¹KEERTHIKA P, ²SURESH P

Assistant Professor (Senior Grade), Department on Computer Science and Engineering
Assistant Professor (Senior Grade), Department on Information Technology
Kongu Engineering College
Perundurai, Erode, Tamilnadu
INDIA

¹keerthikame@gmail.com, ²sureshme@gmail.com

Abstract: - The human civilization advancements lead to complications in science and engineering. Dealing with heterogeneous, geographically distributed resources, grid computing acts as a technology to solve these complicated issues. In grid, scheduling is an important area which needs more focus. This research proposes a hierarchical scheduling algorithm and the factors such as load balancing, fault tolerance and user satisfaction are considered. The proactive fault tolerant approach used here achieves better hit rate. The hierarchical scheduling methodology proposed here results in reduced communication overhead and the user deadline based scheduling results in better user satisfaction when compared to the algorithms which are proposed recent based on these factors. The tool used to evaluate the efficiency of this hierarchical algorithm with other existing algorithms is gridsim. The overall system performance is measured using makespan and it proves to be better for the proposed hierarchical approach.

Key-Words: - Communication overhead, Resource utilization, Load balancing, Fault tolerance, Hierarchical scheduling, User satisfaction.

1 Introduction

Grid computing is a computing paradigm developed to meet the ever increasing computational demands of many applications with increasing number of processors. Grid can be of two types based on its functionality – computational grids and data grids. A computational grid is a hardware and software infrastructure that provides dependable, pervasive, consistent and inexpensive access to high end computational capabilities. Computational grids are accessible to their users via single interface. They merge extremely heterogeneous resources into a single virtual resource. When data storage is taken as the requirement for establishment of grid, it is termed as data grid, used for data intensive applications which need access, transfer and modification of datasets. Some of the characteristics of computational grid fall as heterogeneity, dynamicity, scalability and reliability.

The components of a grid system are scheduler, load balancer, grid broker and portals. The scheduler is responsible for management and

allocation of tasks to the fittest resource, partitioning of tasks in order to schedule parallel execution. The load balancer is another component that is responsible for workload distribution in a balanced manner and this should always be considered to avoid over commitment of resources. The resource broker pairs services between service provider and service requester. Scheduling can be static or dynamic. Dynamic scheduling can be explained in terms of task execution of resources. Another way of categorizing the scheduling algorithm based on their resource management are centralized scheduling, decentralized scheduling and hierarchical scheduling. In decentralized scheduling, there is no central entity control on resources. Local schedulers play a vital role in scheduling. In centralized scheduling, a central entity is responsible for maintaining and scheduling the resources. This suffers with single point of failure and low scalability. This type of scheduling is not appropriate for large scale grids. Since the control over the resources is more, this is more efficient when compared with decentralized

scheduling. In hierarchical scheduling, different schedulers coordinate at certain level. This type of scheduling lacks in fault tolerance but highly fault tolerant than centralized scheduling. The modes of scheduling are batch mode and immediate mode. In immediate mode scheduling, a task is scheduled as soon as it enters the system. In batch mode scheduling, tasks that enters the system are grouped in batches and scheduled at certain time intervals. The work proposed in this research is an initiative to develop an efficient fault tolerant load balanced algorithm with user satisfaction. Currently there are many scheduling algorithms that deals with user satisfaction, fault tolerance, load balancing separately. Till now there is no such algorithm that serves combined for all these factors which are very much essential for a better scheduling. The proposed algorithm combines all these factors and proves its efficiency.

2 Literature Survey

The grid environment consists of dynamic and heterogeneous resources. These resources changes with time i.e., any new resource can join or any of the old resources can exit the grid environment at any time. Due to uneven job arrival patterns and unequal computing capabilities, some resources in the grid environment get overloaded or some resources get under loaded or some resources remain idle. The occurrences of resource failures are high due to the resource characteristics. Both load balancing and resource failures degrade the system performance and also user satisfaction.

A dynamic, distributed load balancing scheme for a grid environment which provides deadline control for tasks is proposed in [1]. The grid broker assigns gridlets between the resources based on the deadline request. Periodically the resources check their state and make a request to the Grid Broker according to the change of state in load. Then, the Grid Broker assigns Gridlets between resources and scheduling for load balancing under the deadline request.

A new grouping based scheduling algorithm that takes user satisfaction into account is proposed in [2]. In this approach, grouping of fine grained jobs to coarse grained jobs and scheduling those coarse grained jobs based on the deadline is done.

A dynamic load balancing mechanism proposed in [3], provides application level load balancing for individual parallel jobs. It ensures that all loads submitted through the dynamic load balancing environment are distributed in such a way that the overall load in the system is balanced and application programs get maximum benefit from available resources.

A layered load balancing algorithm based on the tree model representation is proposed in [4]. This model consists of three main features: (i) it is layered (ii) It supports heterogeneity and scalability (iii) It is totally independent of any physical architecture of grid. The neighbourhood load balancing strategy is used to decrease the amount of messages exchanged between grid resources. As a consequence, the communication overhead induced by task transfer and workload information flow is reduced, leading to a high improvement in the global throughput of a grid.

The load balancing mechanism for optimal load distribution in a non-dedicated cluster or grid computing system with heterogeneous servers processing both generic and dedicated applications was proposed in [5]. Since each dedicated task has a designated server, load distribution is only applied to generic tasks. So, the goal of load balancing mechanism is to find an optimal load distribution strategy for generic tasks on heterogeneous servers preloaded by different amount of dedicated tasks such that the overall average response time of generic applications is minimized.

An echo system which creates ants on demand to achieve load balancing during their adaptive lives is proposed in [6]. They may bear offspring when they sense that the system is drastically unbalanced and commit suicide when they detect equilibrium in the environment. These ants care for every node visited during their steps and record node specifications for future decision making.

A hybrid load balancing algorithm is proposed in [7]. The tasks will be placed in a task queue. The instantaneous scheme, the First-Come-First-Served (FCFS) of the hybrid scheduler functions to find the earliest completion time of each task individually. If the system workload grows heavy, (i.e., more tasks are waiting in the queue) then the scheduler a chance to performs load balancing. Then the tasks are shifted to other system so that the overloaded condition can be avoided. A system level

load balancing is proposed in [8] which are two folds: First, a distributed load balancing model, transforming any grid topology into a forest structure. Second, a two level strategy is proposed to balance the load among resources of computational grid.

A hybrid load balancing policy is proposed in [9] which integrate static and dynamic load balancing technologies. Essentially, a static load balancing policy is applied to select effective and suitable node sets. This will lower the unbalanced load probability caused by assigning tasks to ineffective nodes. When a node reveals the possible inability to continue providing resources, the dynamic load balancing policy will determine whether the node in question is ineffective to provide load assignment. The system will then obtain a new replacement node within a short time, to maintain system execution performance.

A Dynamic Load Balancing Algorithm based on resource type policy [10]. This algorithm makes changes to the distribution of work among workstations at run-time; it uses current or recent load information when making distribution decisions. Multi computers with dynamic load balancing allocate/reallocate resources at runtime based on a priori task information, which may determine when and whose tasks can be migrated. A dynamic and a distributed protocol is proposed in [11]. The grid is partitioned into a number of clusters. Each cluster has a coordinator to perform local load balancing decisions and also to communicate with other cluster coordinators across the grid to provide inter-cluster load transfers. The distributed protocol uses the clusters of the Grid to perform local load balancing decision within the clusters and if this is not possible, load balancing is performed among the clusters under the control of cluster heads called the coordinators.

A Load Balanced Min-Min algorithm is proposed in [12] which reduces the makespan and increases the resource utilization. This algorithm consists of two-phases. In the first phase, the Min-Min algorithm is executed and in the second phase, the tasks in the overloaded resources are rescheduled to use the unutilized resources. A load balancing approach based on Enhanced GridSim architecture is proposed in [13]. The Machine entity in GridSim 4.0 is treated as a dump entity object in and is not able to participate in any decision making activities. In Enhanced GridSim architecture, the Machine Entity is made as active so, it participates in load balancing.

The grid environment is considered as three levels:- Resource Broker level, Machine level, Processing Entity level. Grid Broker is the top manager of a grid environment which is responsible for maintaining the overall grid activities of scheduling and rescheduling. It gets the information of the work load from grid resources. It sends the tasks to resources for optimization of load. Resource is next to grid Broker in the hierarchy. It is responsible for maintaining the scheduling and load balancing of its machines. Also, it sends an event to grid broker if it is overloaded. Machine is a Processing Entity (PE) manager. It is responsible for task scheduling and load balancing of its PEs. Also, it sends an event to resource if it is overloaded. When a new job arrives at a machine, it submits it to a PE, which is lightly loaded. Before submitting the job, the expected status of the PE after the job submission is predicted. If the submission of the job turns the underloaded PE to overloaded PE then the job is assigned to some other underloaded PE, which may not become overloaded due to its submission. If any of the PE is overloaded, then the few tasks in overloaded PE are shifted to other underloaded PE to avoid the overloaded condition. By this way the load is balanced at Machine level. The same procedure is followed at Resource level and Broker level to balance the load in the grid environment.

A fault tolerant hybrid load balancing algorithm is proposed in [14]. This algorithm is carried out in two phases: Static load balancing and dynamic load balancing. In the first phase, a static load balancing policy selects the desired effective sites to carry out the submitted job. If any of the sites is unable to complete the assigned job, a new site will be located using the dynamic load balancing policy. The assignment of jobs must be adjusted dynamically in accordance with the variation of site status.

A load balancing mechanism, which is proposed in [15], works in 2 phases: In the first phase, job allocation is done based on a defined criterion i.e., the heuristic begins with the set of all unmapped tasks. Then the set of minimum completion times is found, like Min-min heuristic. In second phase, heuristic algorithm works based on machines workload, which consists of 2 steps. In the first step, for each task the minimum, second minimum completion time and minimum execution time are found. Then the difference between these two minimum completion time values is multiplied by the amount of minimum completion time and then

divided by minimum execution time. In the second step, if the number of the remaining tasks is not less than threshold, then the heuristic algorithm is executed to balance the load. Finally, the task which has the criteria value as maximum will be selected and removed from the set of unmapped tasks.

An Augmenting Hierarchical Load Balancing algorithm is proposed in [16]. To evaluate the load of the cluster, probability of deviation of average system load from average load of cluster is calculated and checked for the confinement within a defined range of 0 to 1. The fittest resources are allocated to the jobs by comparing the expected computing power of the jobs with the average computing power of the clusters.

A Failure Detection Service (FDS) mechanism and a flexible failure handling framework is proposed in [17]. The FDS enables the detection of both task crashes and user-defined exceptions. The Grid-WFS is built on top of FDS, which allows users to achieve failure recovery in a variety of ways depending on the requirements and constraints of their applications. The resources are modeled based on the system reliability. Reliability of a grid computing resource is measured by mean time to failure (MTTF), the average time that the grid resource operates without failure. Mean time to repair (MTTR) is the average time it takes to repair the Grid computing resource after failure. The MTTR measures the downtime of the computing resource.

Various fault recovery mechanisms such as checkpointing, replication and rescheduling are discussed in [18]. Taking checkpoints is the process of periodically saving the state of a running process to durable storage. This allows a process that fails to be restarted from the point its state was last saved, or its checkpoint on a different resource. Replication: Replication means maintaining a sufficient number of replicas, or copies, of a process executing in parallel on different resources so that at least one replica succeeds.

In [19], it is described that the fault tolerance is an important property in order to achieve reliability. Reliability indicates that a system can run continuously without failure. A highly reliable system is the one that continues to work without any interruption over a relatively long period of time. The fault tolerance is closely related to Mean Time to Failure (MTTF) and Mean Time between Failures (MTBF). MTTF is the average time the system operates until a failure occurs, whereas the MTBF is

the average time between two consecutive failures. The difference between the two is due to the time needed to repair the system following the first failure. Denoting the Mean Time to Repair by MTTR, the MTBF can be obtained as $MTBF = MTTF + MTTR$.

A check pointing mechanism is proposed in [20] to achieve fault tolerance. The check pointing process periodically saves the state of a process running on a computing resource so that, in the event of resource failure, it can resume on a different resource. If any resource failure happens, it invokes the necessary replicas in order to meet the user application reliability requirements.

In our previous work [21], we have proposed an efficient fault tolerant scheduling algorithm (FTMM) which is based on data transfer time and failure rate. System performance is also achieved by reducing the idle time of the resources and distributing the unmapped tasks equally among the available resources. A scheduling strategy that considers user deadline and communication time for data intensive tasks with reduced makespan, high hit rate and reduced communication overhead is introduced in [22]. This strategy does not consider the occurrence of resource failure.

In our previous work [23], we have proposed a new Bicriteria scheduling algorithm that considers both user satisfaction and fault tolerance. The proactive fault tolerant technique is adopted and the scheduling is carried out by considering the deadline of gridlets submitted. The main contribution of this paper includes achieving user satisfaction along with fault tolerance and minimizing the makespan of jobs. In our previous work [24], we have proposed a multi-criteria scheduling algorithm that considers load balancing, fault tolerance and user satisfaction as a centralized approach.

A Prioritized user demand algorithm is proposed in [25] that considers user deadline for allocating jobs to different heterogeneous resources from different administrative domains. It produces better makespan and more user satisfaction but data requirement is not considered. While scheduling the jobs, failure rate is not considered. So the scheduled jobs may be failed during execution. A work based on user satisfaction and hierarchical load balancing is proposed in [26] that considers user demands and 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 main contribution in this work is that a hierarchical scheduling algorithm is proposed which considers multiple constraints such as user deadline, failure rate, load of resources and communication overhead at the time of scheduling.

3 Materials and Methods

3.1 Problem Formulation

We have proposed a scheduling architecture in our previous work which is for centralized scheduling of resources. In this work, a hierarchical scheduling architecture given below in figure 1 is followed. The proposed hierarchical scheduling algorithm is static following a batch mode of scheduling.

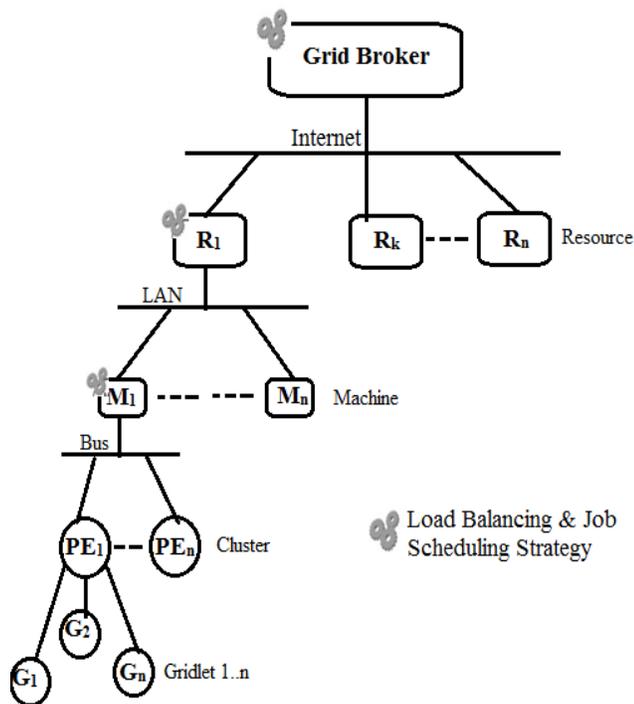


Fig.1 Hierarchical Scheduling Architecture

3.2 Proposed Hierarchical Algorithm

In this work, a hierarchical scheduling methodology is proposed. The tasks are expected to be submitted by the user at different levels of hierarchy such as machine, resource, grid broker. A machine is a collection of processing elements (PE's). A resource is a collection of machines and a grid broker is a collection of resources that has all the information about the resources such as capacity, availability etc. The hierarchy followed here is similar to the hierarchy followed by gridsim which is given below in figure 2.

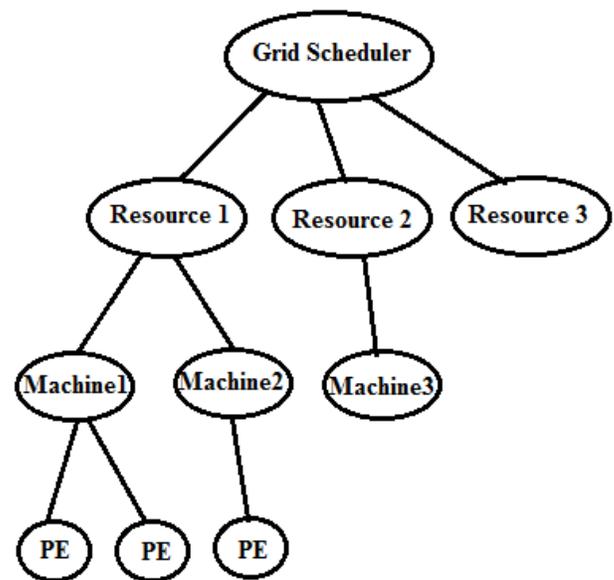


Fig.2 GridSim Architecture

The scheduling algorithm takes load of PE's, average load of machines, average load of resources and average load of the system as factors in deciding the applicable resource/machine/PE in order to balance the load of the grid system. The factors such as user deadline of tasks submitted by the user are considered at the time of scheduling in order to achieve user satisfaction. The calculation of load of each PE, machine and resource becomes essential since scheduling is carried out at three levels. Load of each PE is calculated using the formula

$$Load(PE_i) = \frac{\sum_{j=0}^m MI_j}{MIPS_i \times AT_i} \quad (1)$$

where m is the number of tasks allocated to PE_i and AT_i is the availability time of PE_i . Load of each Machine is calculated using the formula

$$Load(M_i) = \sum_{j=0}^n Load(PE_j) \quad (2)$$

where j is the number of tasks submitted to PEs under that machine. Load of each resource is calculated using the formula

$$Load(R_i) = \sum_{j=0}^n Load(M_j) \quad (3)$$

where j is the number of tasks submitted to PEs under that resource. The average load of each machine is calculated by,

$$AL(M_i) = \frac{\sum_{k=1}^n Load(PE_k)}{n} \quad (4)$$

The average load of each resource is calculated by,

$$AL(R_i) = \frac{\sum_{k=1}^n AL(M_k)}{n} \quad (5)$$

and the average load of grid broker is calculated by,

$$AL(GB_i) = \frac{\sum_{k=1}^n AL(R_k)}{n} \quad (6)$$

After calculating the load and average load, the balance threshold is calculated in order to categorize the resources as overloaded, underloaded and normally loaded. The balance threshold is calculated at machine level by using the formula,

$$\Omega_M = AL(M_i) + \sigma_M \quad (7)$$

At resource level, the balance threshold is calculated as,

$$\Omega_R = AL(R_i) + \sigma_R \quad (8)$$

The balance threshold at grid broker level is calculated as,

$$\Omega_{GB} = AL(GB_i) + \sigma_{GB} \quad (9)$$

where σ_M , σ_R , σ_{GB} are the deviation factors at machine, resource and grid broker levels respectively. At machine level, the deviation factor can be calculated by,

$$\sigma_M = \sqrt{\frac{\sum_{i=1}^N (Load(PE_i) - AL(M_i))^2}{N}} \quad (10)$$

At resource level, the deviation factor is calculated by,

$$\sigma_R = \sqrt{\frac{\sum_{i=1}^N (Load(M_i) - AL(R_i))^2}{N}} \quad (11)$$

and at the grid broker level, the deviation factor is given by,

$$\sigma_{GB} = \sqrt{\frac{\sum_{i=1}^N (Load(R_i) - AL(GB_i))^2}{N}} \quad (12)$$

The hierarchical scheduling algorithm is given in Algorithm 1. It works as follows. If a task is submitted by the user at machine level which is the lower level in the hierarchy, then the scheduling algorithm works for the PE's under that machine. Whenever a task is submitted at resource level, they are scheduled to the PE's under the machines which are under that particular resource. When a task is submitted at the grid broker level which is the higher level of hierarchy, they are scheduled to the resources under that grid broker. If the tasks at any level is failed to be scheduled, then it is sent to its higher level of the hierarchy and scheduled. The balance threshold is very important in deciding whether a resource is over loaded, under loaded or normally loaded. After categorizing the resources, the resources which are underloaded are considered for scheduling. This is illustrated with equations 7, 8, 9, 10, 11 and 12.

For all tasks MT_i submitted at Machine level,
 Perform possible allocation to the tasks in the list of PE's under that machine using algorithm 2.
 If MT_i is not empty,
 Submit the tasks in MT_i list to the next upper level of the hierarchy RT_i
 For all tasks RT_i submitted at Resource level,
 Perform possible allocation to the tasks in the list of PE's under that resource using algorithm 3.
 If RT_i is not empty,
 Submit the tasks in RT_i list to the next upper level of the hierarchy GBT_i
 For all tasks GBT_i submitted at Grid Broker level,
 Perform possible allocation to the tasks in GBT_i to the list of PE's under that Grid Broker using
 algorithm 4.
 If GBT_i is not empty,
 Increment J_f which is the number of tasks not scheduled.

Algorithm 1: Hierarchical Scheduling Algorithm

1. Get the list of tasks MT_i with their user deadline UD_i .
2. Get the list of PE's under that machine from GIS.
3. Construct $ETC(T_i, R_j)$ matrix of size $m \times n$ where m is the number of tasks and n is the number of PE's under that machine where the tasks are submitted.
4. For all PE_j in the list,
 - 4.1 Calculate failure rate $FR(R_j)$ where R_j represents PE_j
 - 4.2 Calculate $RT(R_j)$ is the number of tasks submitted to R_j .
 - 4.3 Calculate Load of PE's, Average load of machine.
5. Calculate balance threshold at machine level
6. Create a list of underloaded PE's (UP) which has $Load(PE_i) < \Omega_M$.
7. For each task T_i in MT_i in queue and for each PE_j ,
 Construct $CT(T_i, R_j)$, $DT(T_i, R_j)$, $TCT(T_i, R_j)$ matrix of size $m \times n$
8. For all task T_i in MT_i
 - 8.1 Create list UT_{i_1} and UT_{i_2} with PE's that has $TCT(T_i, R_j) \leq UD_{T_i}$ and $TCT(T_i, R_j) > UD_{T_i}$ respectively.
 - 8.2 Sort UT_{i_1} and UT_{i_2} based on $FR(R_j)$ of resources in ascending order
 - 8.3 Create lists ULT_{i_1} and ULT_{i_2} with the set of underloaded resources from UT_{i_1} and UT_{i_2} respectively in order.
 - 8.4 If entries in ULT_{i_1} ,
 Select the first resource in the list for task T_i and dispatch T_i to resource R_j and Increment Deadline Hit Count and Hit Count.
 else if entries in ULT_{i_2} ,
 Select the first resource in the list for task T_i and dispatch T_i to resource R_j and Increment Hit Count.
 - 8.5 Remove task T_i from Task_list MT_i .
 - 8.6 Update $RT(R_j)$ and $FR(R_j)$ where j is the PE to which the task T_i is dispatched.
9. If there are tasks in Task_list T ,
 Repeat steps from 4.3.
 Endif

Algorithm 2: Scheduling at Machine Level

The scheduling algorithm at the machine level is given in algorithm 2. The algorithm works as follows. The machine receives the tasks with user deadline $UD(T_i)$. The task's information such as its length in MI is used to calculate the execution time $ETC(T_i, R_j)$ of each task in each of the available resources. With the ready time information $RT(R_j)$ available for each resource at GIS, the algorithm calculates the completion time $CT(T_i, R_j)$. The failure information of resources such as number of tasks submitted to a resource T_{sub} and number of tasks successfully completed T_{succ} and number of tasks not completed successfully T_f is also available in GIS which helps in calculating the failure rate $FR(R_j)$.

The list of resource in which the task gets completed within user deadline is collected for each task and they are sorted based on their failure rate. Based on the balance threshold, the resources are categorized as overloaded and underloaded and finally the load is balanced by submitting the task to the underloaded resource. When a resource is assigned a task, the load of each resource and system, balance threshold, failure rate and ready time are recalculated. The same procedure is repeated for all tasks till the task list becomes empty.

The scheduling algorithm at resource level and grid broker level is given in algorithm 3 and 4 respectively.

4 Results and Discussion

4.1 Experimental Environment

The simulation is based on the scheduling architecture in Figure 1. The number of PE's and tasks considered is 16 and 512 respectively. The number of machines ranges from 1 to 4 and number of PEs per machine ranges from 1 to 2. The 16 PE's are grouped to form number of machines and machines are grouped to form resources which are again grouped and controlled by resource/grid broker.

The factors considered in designing this algorithm are user satisfaction, which can be evaluated using deadline hit count, fault tolerance, which can be evaluated using hit count, load balancing, which can be evaluated using average

resource utilization and when these are applied in a hierarchical manner, it is evaluated using communication time and as a whole the system performance is improved, which can be evaluated using makespan. The performance metrics such as makespan, hit count, deadline hit count and average resource utilization are defined below.

Makespan: Makespan is one of the most important standard metric of grid scheduling to measure its performance. It is defined as the overall completion time of a batch of tasks and is given by,

$$\text{Makespan} = \max\{RT(R_j)\}, \forall j \in n \quad (13)$$

It is used to measure the ability of grid to accommodate gridlets in less time.

Hit count: Hit count is a new metric introduced in this chapter. It represents the number of tasks successfully completed in a batch of tasks. Here, each batch is assumed to have 512 tasks and the hit count gives the number of tasks successfully completed out of 512.

Deadline Hit Count: This is a new metric introduced in this chapter which represents the number of tasks successfully completed within the given user deadline.

Average resource utilization: This metric is newly introduced in order to measure the load balancing which can be calculated as follows.

The utilization of each resource $RU(R_j)$ can be calculated by the Equation (14).

$$RU(R_j) = \frac{\sum_{i=0}^m MI_i}{MIPS_j \times AT_j} \times 100 \quad (14)$$

The average resource utilization ARU of the system can be calculated using Equation (15).

$$ARU = \frac{1}{N} \sum_{j=1}^N RU(R_j) \quad (15)$$

where N is the number of resources.

Communication Time: In addition to these metrics, this chapter introduces a new metric named communication time which is the time taken for transferring the tasks between different levels of hierarchy.

1. Get the list of tasks RT_i with their user deadline UD_i .
2. Get the list of PE's under that resource from GIS.
3. Construct $ETC(T_i, R_j)$ matrix of size $m \times n$ where m is the number of tasks and n is the number of PE's under that resource where the tasks are submitted.
4. For all PE_j in the list,
 - Do
 - 4.1 Calculate failure rate $FR(R_j)$, where R_j represents PE_j
 - 4.2 Calculate $RT(R_j)$ where n is the number of tasks submitted to R_j .
 - 4.3 Calculate Load of machine, Average load of resource.
 - Done
5. Calculate balance threshold at resource level
6. Create a list of underloaded machine's (UM) which has $Load(M_i) < \Omega_R$.
7. For each task T_i in RT_i in queue and for each PE_j ,
 - Do
 - Construct $CT(T_i, R_j), DT(T_i, R_j), TCT(T_i, R_j)$ matrix of size $m \times n$
 - Done
8. For all task T_i in RT_i
 - Do
 - 8.1 Create list UT_{i_1} and UT_{i_2} with PE's that has $TCT(T_i, R_j) \leq UD_{T_i}$ and $TCT(T_i, R_j) > UD_{T_i}$ respectively.
 - 8.2 Sort UT_{i_1} and UT_{i_2} based on $FR(R_j)$ of resources in ascending order
 - 8.3 Create lists ULT_{i_1} and ULT_{i_2} with the set of underloaded resources from UT_{i_1} and UT_{i_2} respectively in order.
 - 8.4 If entries in ULT_{i_1} ,
 - Select the first resource in the list for task T_i and dispatch T_i to resource R_j and Increment Deadline Hit Count and Hit Count.
 - else if entries in ULT_{i_2} ,
 - Select the first resource in the list for task T_i and dispatch T_i to resource R_j and Increment Hit Count.
 - 8.5 Remove task T_i from Task_list RT_i .
 - 8.6 Update $RT(R_j)$ and $FR(R_j)$ where j is the resource to which the task T_i is dispatched.
 - Done
9. If there are tasks in Task_list T ,
 - Repeat steps from 4.3.
- Endif

Algorithm 3: Scheduling at Resource Level

1. Get the list of tasks GBT_i with their user deadline UD_i .
 2. Get the list of PE's under that grid broker from GIS
 3. Construct $ETC(T_i, R_j)$ matrix of size $m \times n$ where m is the number of tasks and n is the number of PE's under that grid broker where the tasks are submitted.
 4. For all PE_j in the list,
 - Do
 - 4.1 Calculate failure rate $FR(R_j)$, where R_j represents PE_j
 - 4.2 Calculate $RT(R_j)$ where n is the number of tasks submitted to R_j .
 - 4.3 Calculate Load of resource, Average load of grid broker
 - Done
 5. Calculate balance threshold at grid broker level
 6. Create a list of underloaded resource's (UR) which has $Load(R_i) < \Omega_{GB}$.
 7. For each task T_i in GBT_i in queue and for each PE_j ,
 - Do
 - Construct $CT(T_i, R_j), DT(T_i, R_j), TCT(T_i, R_j)$ matrix of size $m \times n$
 - Done
 8. For all task T_i in GBT_i
 - Do
 - 8.1 Create list UT_{i_1} and UT_{i_2} with PE's that has $TCT(T_i, R_j) \leq UD_{T_i}$ and $TCT(T_i, R_j) > UD_{T_i}$ respectively.
 - 8.2 Sort UT_{i_1} and UT_{i_2} based on $FR(R_j)$ of resources in ascending order
 - 8.3 Create lists ULT_{i_1} and ULT_{i_2} with the set of underloaded resources from UT_{i_1} and UT_{i_2} respectively in order.
 - 8.4 If entries in ULT_{i_1} ,
 - Select the first resource in the list for task T_i and dispatch T_i to resource R_j and Increment Deadline Hit Count and Hit Count.
 - else if entries in ULT_{i_2} ,
 - Select the first resource in the list for task T_i and dispatch T_i to resource R_j and Increment Hit Count.
 - 8.5 Remove task T_i from Task_list GBT_i .
 - 8.6 Update $RT(R_j)$ and $FR(R_j)$ where j is the resource to which the task T_i is dispatched.
 - Done
9. If there are tasks in Task_list T,
 - Repeat steps from 4.3.
- Endif

Algorithm 4: Scheduling at Grid Broker Level

4.2 Simulation Results

The HRL_LBFT algorithm is evaluated for the above defined metrics. The results are compared with Min-min, FTMM, BSA, LBFT and LBECS algorithms.

The makespan values of the algorithms such as HRL_LBFT, Min-min, FTMM, BSA, LBFT and LBECS are shown in figure 3. The results show that the proposed HRL_LBFT has minimized makespan than the other algorithms. The Min-min is a benchmark algorithm for measuring the scheduling algorithm's performance based on makespan. But it is noted that the makespan of proposed HRL_LBFT has a notable improvement over Min-min.

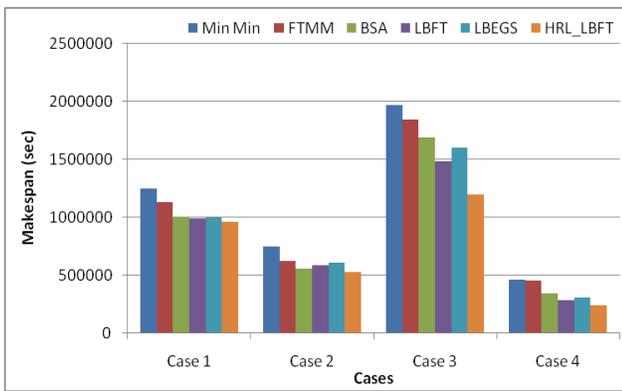


Fig. 3 Performance based on Makespan (sec)

The hit count values of the algorithms such as HRL_LBFT, Min-min, FTMM, BSA, LBFT and LBECS are shown in figure 4. The results show that the proposed HRL_LBFT has highest hit count than the other algorithms. Since LBECS only concentrates on load balancing, the hit count is very low when compared with other algorithms.

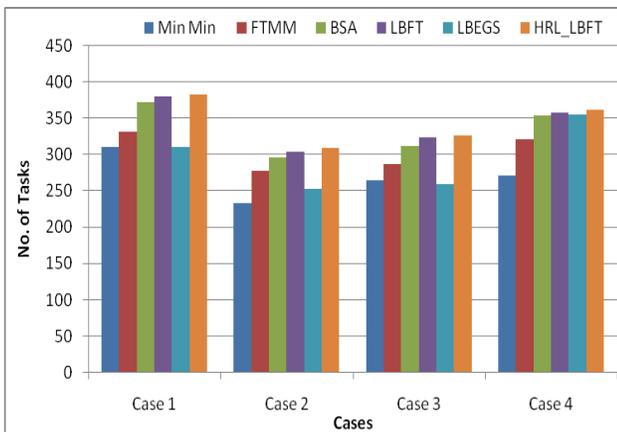


Fig. 4 Performance based on Hit Count

The deadline hit count values of the algorithms such as HRL_LBFT, Min-min, FTMM, BSA, LBFT and LBECS are shown in figure 5. The results show that the proposed HRL_LBFT relatively has a high number of deadline hits than the other algorithms. Since LBECS doesn't concentrate on user satisfaction, its deadline hit count is low when compared to BSA and HRL_LBFT which considers user satisfaction.

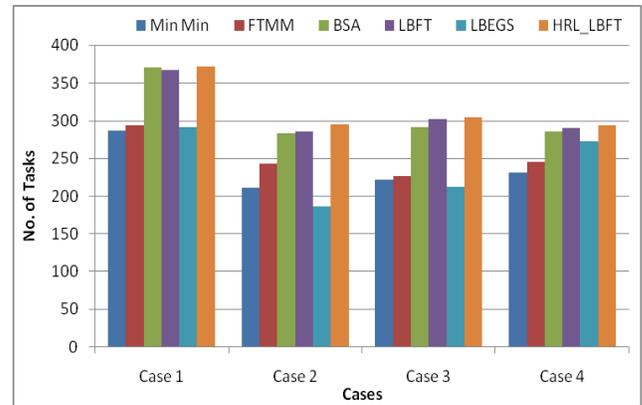


Fig. 5 Performance based on Deadline Hit Count

The average resource utilization of the algorithms such as HRL_LBFT, Min-min, FTMM, BSA, LBFT and LBECS are shown in figure 6 and the results shows that the proposed HRL_LBFT relatively has high resource utilization than Min-min, FTMM and BSA, but nearly same utilization as LBFT algorithm.

Communication time is a measure of overhead due to transfer of tasks from one level of hierarchy to another. The centralized LBFT algorithm and the proposed hierarchical HRL_LBFT algorithms are compared in figure 7 based on this metric in order to prove the efficiency of hierarchical approach. The results show that the hierarchical approach reduces the communication time in a remarkable way.

The average percentage improvement of HRL_LBFT based on makespan towards Min-min is 34.8%. With FTMM, the percentage improvement is 28.3 % and towards BSA, an improvement of 17% is achieved. When compared with LBFT, it shows an average percentage improvement of 12.4% and with LBECS, it shows 16.4% improvement. The average percentage improvement of HRL_LBFT based on hit count towards Min-min is 28.2%. With FTMM, the percentage improvement is 13.2 % and towards BSA, an improvement of 3.5% is achieved. When

compared with LBFT, it shows an average percentage improvement of 1% and with LBEGS, it shows 18.2% improvement.

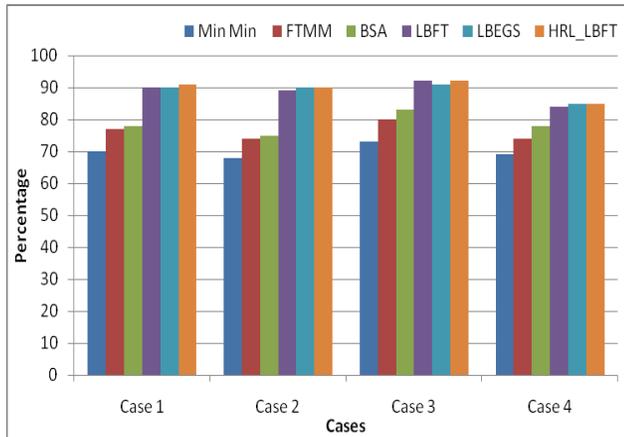


Fig. 6 Performance based on Average Resource Utilization

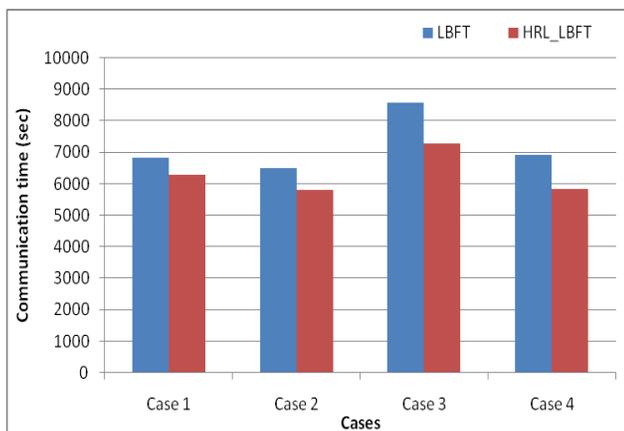


Fig. 7 Performance based on Communication Time (sec)

The average percentage improvement of HRL_LBFT based on deadline hit count towards Min-min is 33.4%. With FTMM, the percentage improvement is 25.6 % and towards BSA, an improvement of 3% is achieved. When compared with LBFT, it shows an average percentage improvement of 1.8% and with LBEGS, it shows 34.2% improvement. Based on resource utilization, the HRL_LBFT algorithm has a percentage improvement of 19.5% over Min-min, 13.3% over FTMM, 11% over BSA, 0.8% over LBFT and 0.5% over LBEGS. Based on communication time,

HRL_LBFT performs better with a percentage improvement of 12.4% over LBFT.

5 Conclusions and Future Work

In this work, a scheduling algorithm taking user satisfaction, load balancing, and fault tolerance into account is proposed. Scheduling is done at three levels such as machine level, resource level and grid broker level. Because of this hierarchy considered, the communication time is minimized that in turn minimizes the makespan. Currently many works are carried out separately for load balancing, fault tolerance and user satisfaction. But this proposed work gives a combined solution with all these factors.

The efficiency of this algorithm is proved with the evaluation parameters such as makespan, deadline hit count, hit rate, resource utilization, average resource utilization and communication overhead. In this work, the tasks considered are computation intensive tasks. In future, data intensive tasks can also be considered for scheduling and this can be again extended with the factor of security in grid which would lead to an efficient work that can be used in computational grid. Also, like user deadline, the budget for task execution can also be obtained as a task requirement from user and scheduling can be done considering it.

References

- [1] Hao Y, Liu G. and Wenc N, An enhanced load balancing mechanism based on deadline control on GridSim, *Future Generation Computer Systems*, Vol.28, 2012, pp. 657-665.
- [2] Suresh P, Balasubramanie P, Grouping based User Demand Aware job scheduling Approach for computational Grid, *International Journal of Engineering Science and Technology*, Vol.4, No.12, 2012, pp.4922-4928.
- [3] Payli R.U., Yilmaz E., Ecer A., Akay H.U. and Chien S., DLB- A Dynamic load balancing tool for grid computing, *The Journal of scalable computing*, Vol.7, No.2, 2006, pp. 15-23.

- [4] Yagoubi B. and Slimani Y., Load balancing strategy in grid environment, *Journal of Information Technology and Applications*, Vol.1, No.4, 2007, pp. 285-296.
- [5] Li K., Optimal load distribution in non dedicated heterogeneous cluster and grid computing environments, *Journal of Systems Architecture*, Vol.54, 2008, pp. 111–123.
- [6] Salehi M.A., Deldari H. and Dorri B.M., Balancing Load in a Computational Grid Applying Adaptive, Intelligent Colonies of Ants, *Informatics*, Vol.32, 2008, pp. 159-167.
- [7] Li Y., Yang Y. and Zhou L., A hybrid load balancing strategy of sequential tasks for grid computing Environments, *Future Generation Computer Systems*, Vol.25, 2009, pp. 819-828.
- [8] Yagoubi B. and Meddeber M., Distributed Load Balancing Model for Grid Computing, *Journal of Computer Science*, Vol.12, 2010, pp. 43-60.
- [9] Yan K.Q., Wang S.S., Wang S.C. and Chang C.P., Towards a hybrid load balancing policy in grid computing system, *Expert Systems with Applications*, Vol.36, 2009, pp. 12054-12064.
- [10] Kumar K., A Dynamic Load Balancing Algorithm in Computational Grid using Fair Scheduling, *International Journal of Computer Science*, Vol.8, No.1, 2011, pp. 123-129.
- [11] Payli R.U., Erciyes K. and Dagdeviren O., Cluster-Based Load Balancing Algorithms for Grids, *International Journal of Computer Networks & Communications*, Vol.3, No.5, 2011, pp. 253-269.
- [12] Kokilavani T., Load Balanced Min-Min Algorithm for Static Meta-Task Scheduling in Grid Computing, *International Journal of Computer Applications*, Vol.20, No.2, 2011, pp. 43-49.
- [13] Qureshi K., Reman A. and Manl P., Enhanced GridSim architecture with load balancing, *The Journal of Supercomputing*, Vol.57, No. 3, 2011, pp. 265-275.
- [14] Balasangameshwara J. and Raju N., A hybrid policy for fault tolerant load balancing in grid computing environments, *Journal of Network and Computer Applications*, Vol.35, 2012, pp. 412-422.
- [15] Bardsiri A.K. and Rafsanjani M.K., A New Heuristic Approach Based on Load Balancing for Grid Scheduling Problem, *Journal of Convergence Information Technology*, Vol.7, No.1, 2012, pp. 329-336.
- [16] Raj J.S., Hridya K.S. and Vasudevan V., Augmenting Hierarchical Load Balancing with Intelligence in Grid Environment, *International Journal of Grid and Distributed Computing*, Vol. 5, No. 2, 2012, pp. 9-18.
- [17] Hwang S. and Kesselman C., A Flexible Framework for Fault Tolerance in the Grid, *Journal of Grid Computing*, Vol.1, 2003, pp. 251-272.
- [18] Dabrowski C., Reliability in grid computing, *International Journal of Computer Science*, Vol.8, No.1, 2009, pp. 123-129.
- [19] Latchoumy P. and Khader S.A.P., Survey on Fault Tolerance in Grid Computing, *International Journal of Computer Science & Engineering Survey (IJCSES)*, Vol.2, No.4, 2011, pp. 97-110.
- [20] Priya B.S., Fault Tolerance and Recovery for Grid Application Reliability using Check Pointing Mechanism, *International Journal of Computer Applications*, Vol.26, No.5, 2011, pp. 32-37.
- [21] Keerthika P, Kasthuri N, An Efficient Fault Tolerant Scheduling Approach for Computational Grid, *American Journal of Applied Sciences*, Vol. 9, Issue 12, 2013, pp. 2046-2051.
doi:10.3844/ajassp.2012.2046.2051.
- [22] Suresh P, Balasubramanie P, User Demand Aware Scheduling Algorithm for Data Intensive Tasks in Grid Environment, *European Journal of Scientific Research*, Vol.74, No.4, 2012, pp.609-616.
- [23] Keerthika P, Kasthuri N, An Efficient Grid Scheduling Algorithm with Fault Tolerance and User Satisfaction, *Mathematical Problems in Engineering*, Volume 2013, Article ID 340294, 2013.
- [24] Keerthika P, Kasthuri N, A Hybrid Scheduling Algorithm with Load Balancing for Computational Grid, *International Journal of Advanced Science and Technology*, Vol. 58, 2013, pp.13-28.

- [25] Suresh P, Balasubramanie P and Keerthika P, Prioritized User Demand Approach for Scheduling Meta Tasks on Heterogeneous Grid Environment, *International Journal of Computer Applications*, Volume 23, No.1, 2011.
- [26] Suresh P, Balasubramanie P, User Demand Aware Grid Scheduling Model with Hierarchical Load Balancing, *Mathematical Problems in Engineering*, Volume 2013, Article ID 439362, 2013.