Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2013, Article ID 340294, 9 pages
http://dx.doi.org/10.1155/2013/340294

Research Article

An Efficient Grid Scheduling Algorithm with Fault Tolerance and User Satisfaction

P. Keerthika[1] and N. Kasthuri[2]

[1] *Department of CSE, Kongu Engineering College, Perundurai, Erode, Tamilnadu 638052, India*
[2] *Department of ECE, Kongu Engineering College, Perundurai, Erode, Tamilnadu 638052, India*

Correspondence should be addressed to P. Keerthika; keerthikame@gmail.com

Received 8 January 2013; Revised 1 April 2013; Accepted 2 April 2013

Academic Editor: Engang Tian

Problem Statement. The advances in human civilization lead to more complications in problem solving. Grid computing serves as an efficient technology in solving those complicated problems. In computational grids, the grid scheduler schedules the task and finds the appropriate resource for each task. The scheduler must consider several factors such as user demand, communication time, failure handling mechanisms, and reduced makespan. Most of the existing algorithms do not consider user satisfaction. Thus a scheduling algorithm that handles failure of resources and achieves user satisfaction gains more importance. *Approach*. A new bicriteria scheduling algorithm (BSA) that considers user satisfaction along with fault tolerance has been introduced. The main contribution of this paper includes achieving user satisfaction along with fault tolerance and minimizing the makespan of jobs. *Results*. The performance of this proposed algorithm is evaluated using GridSim based on makespan and number of jobs completed successfully within user deadline. *Conclusions/Recommendations*. The proposed BSA algorithm achieves reduced makespan and better hit rate with higher user satisfaction and fault tolerance.

1. Introduction

Grid computing is a collection of dynamic and heterogeneous resources from different administrative domains and provides accessing of the resources. The resources in the grid are grouped together to form a virtual organization that is applied to solve a large scientific or business problem. The main aim of this technology is to share idle and scattered resources such as computational power and storage capacity. Grid computing is mainly divided into two types. They are computational grid and data grid. Computational grid is defined as combination of hardware and software infrastructure that provides consistent, pervasive, and inexpensive access to high end computational resources. Data grid is a combination of large datasets that are mainly used to provide data for applications.

In order to utilize the resources efficiently and to satisfy all the requirements of the users, there is a need for effective scheduling algorithm. Grid scheduler mainly deals with how efficiently and appropriately to assign resources to jobs. While choosing resources for jobs, scheduler should consider various characteristics of jobs such as length of the job, user deadline, and resource characteristics such as capability, communication time, and cost to execute the jobs.

Grid scheduler works in three phases: resource discovery phase, and resource allocation phase and job execution phase. Resource discovery phase involves identifying the available resources from the resource pool, whereas resource allocation phase involves selection of suitable resources and allocating the selected resources to the jobs. The third phase is executing the jobs at resource locations. Grid scheduler searches the fittest resource for a job along with minimized makespan or execution time, best utilization of the resources, and improved user satisfaction.

Many scheduling algorithms have been developed in the last decade. These scheduling algorithms mainly concentrate on reducing makespan and fault tolerance and some algorithms concentrate on user deadline. The scope of this work is extended to user satisfaction by considering the deadline of

each job along with grid reliability which is very much required to provide correct service. This section describes some of the scheduling algorithms.

He et al. [1] proposed an adaptive scheduling algorithm that considers both quality of service (QoS) and nondedicated computing. Similar to the existing task scheduling algorithms, this scheduling algorithm is designed to achieve high throughput computing.

A minimum time to release job scheduling algorithm is proposed by Malarvizhi and Uthariaraj [2] in which time to release (TTR) is calculated. The tasks are arranged in descending order based on TTR value. Tasks are scheduled in the sorted order. This algorithm performs better when compared with First-Come-First-Serve and Min-Min algorithms. Suresh and Balasubramanie [3] proposed a group-based scheduling algorithm which considers user deadline and reduces communication overhead by adopting the grouping technique.

Buyya et al. [4] described a cost optimization scheduling algorithm to optimize the cost to execute the jobs. It also reduces the execution time of the jobs. But in this algorithm, failure rate of the resources and user deadline of the jobs are not considered. Zheng et al. [5] proposed a fault-tolerant scheduling for differentiated classes of independent tasks. This methodology has two algorithms such as MRC-ECT and MCT-LRC based on minimum replication cost and minimum completion time, respectively. This algorithm does not consider the fault rate of the computational resources.

Lee et al. [6] proposed a fault-tolerance-based resource management service which considers different types of failure and QoS requirement. This algorithm fails to concentrate on user satisfaction and execution time. A greedy metascheduling algorithm based on multiple simultaneous requests is proposed by Subramani et al. [7]. In this algorithm, scheduler identified the sites that can start the job earliest. This is suitable only for homogeneous resources and does not take data requirements into account.

Lin et al. [8] proposed an application demand aware algorithm which considers application demand of the jobs for scheduling. It produces better user satisfaction and fault rate of the resources is not considered. A prioritized user demand algorithm is proposed by Suresh et al. [9] 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.

Benoit et al. [10] proposed a novel method of modelling job execution on grid compute clusters. This algorithm uses Performance Evaluation Process Algebra (PEPA) as the system description formalism, capturing both workload and computing fabric.

In our previous work [11], 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 by Suresh and Balasubramanie [12]. This strategy does not consider the occurrence of resource failure.

The fault tolerant algorithm discussed by Garg and Singh [13] surveys the importance of fault tolerance for achieving reliability by all possible mechanisms such as replication, check pointing, and job migration. It extends the cost-optimisation algorithm to optimise the time without incurring addi tional processing expenses. This is accomplished by applying the time-optimisation algorithm to schedule task farming or parameter-sweep application jobs on distributed resources having the same processing cost. Modiri et al. [14] use DAG mechanism to enter tasks and thereby bring out an efficient algorithm, namely, ant colony optimization algorithm.

In the literature, Medeiros et al. [15], fault tolerance in grid environment can be divided into proactive and postactive mechanisms. The pro-active mechanisms consider the job failure history of each resource before scheduling of a job and dispatche jobs to resources with hopes that the job does not fail, whereas post-active mechanisms handle the job failure after it has occurred. However, for dynamic grid systems only post-active mechanism is more relevant than the pro-active mechanisms.

For grid environment, Wrzesińska et al. [16] proposed that there are several reasons for workflow execution failure. The reasons are variation in the execution environment configuration, nonavailability of required services or software components, overloaded resource conditions, system running out of memory, and faults in computational and network fabric components. Grid workflow management systems should be able to identify and handle failures and support reliable execution in the presence of concurrency and failures.

At the application level and for resource management systems, many fault-tolerance techniques are proposed for cluster and grid environments which are discussed by Li et al. [17] and Limaye et al. [18]. Since the grid infrastructure is still emerging, there are only few works that have been done for job failure analysis and they all find it tough to collect traces at hierarchical levels.

Favarim et al. [19] introduced a new fault tolerance approach with several masters called brokers. The grid brokers receive jobs from their users and divide them into tasks. These subtasks are made available to the resources that compose the grid. Brokers are usually specific to one class of applications and they only know how to decompose jobs of this class. For example, if the application deals with processing satellite images, the images are treated as jobs and the broker decomposes the job into several tasks and analyzes them by different resources. After executing a task, the resources send the result to the broker and the broker assembles all results and returns them to the user. Favarim et al. [20] propose that all communication between brokers/masters and resources/workers is done exclusively through the tuple space.

The survey done by Christopher [21] shows that both the commercial grid systems and research grid systems that are currently in use behave reliably at present levels of scale using available technology. However, efforts to develop reliable and

fault tolerant methods for grid environments are in progress with increased scale, heterogeneity, and dynamism.

The challenge of ensuring reliability in grid systems is discussed by Foster et al. [22–24]. The vision of grid systems was articulated in which computing and data resources belonging to many enterprisers are organized into a single, virtual computing entity that can be transparently utilized to solve compute- and data-intensive problems. Subsequently, this vision has continued to evolve as use of grid technology has grown within industry and science.

The main objective of this paper is to develop a scheduling algorithm which reduces makespan and idle time of the resources. It also ensures that the tasks are completed within the user expected deadline. While scheduling the jobs, failure rate of the resources is also considered. The proposed algorithm improves system performance and user satisfaction and reduces number of failures of the jobs by considering fault rate of the resources.

The remaining part of this paper is organized as follows. Section 2 describes the problem formulation with the proposed scheduling architecture and the proposed algorithm. The results obtained for the parameters considered are compared with the Min-Min algorithm and FTMM algorithm in Section 3.

2. Materials and Methods

2.1. Problem Formulation. Grid scheduling is an important aspect of computational grid. Scheduler plays a major role in scheduling the submitted tasks based on their requirements. The scheduling architecture is given in Figure 1 which has a scheduler where the proposed BSA scheduling algorithm works.

The grid information service (GIS) holds the information of all the resources such as resource id, resource availability, resource capacity, type of resource, and current load of the resource. The users submit the tasks to the grid scheduler and the grid scheduler assigns the submitted tasks to the available resources based on their requirements.

Each resource differs from other resources in many ways that include number of processing elements, processing speed, internal scheduling policy, its load factor, and so forth. Similarly, each job differs from other jobs by execution time, deadline, time zone, and so forth. The static mapping of metatasks is done in which each machine executes one task at a time. It is assumed that the size of the metatasks, number of resources, and expected execution time of each task in each machine are known priori.

An ETC matrix (expected time to compute) is constructed using the EET which is the estimated execution time of task i on resource j. The experimental results are based on Braun et al. [25] wherein the scheduling problem is defined by

(i) a number of independent tasks to be allocated to the available grid resources,

(ii) number of resources that are available to participate in the allocation of tasks,

(iii) workload of each task (MI),

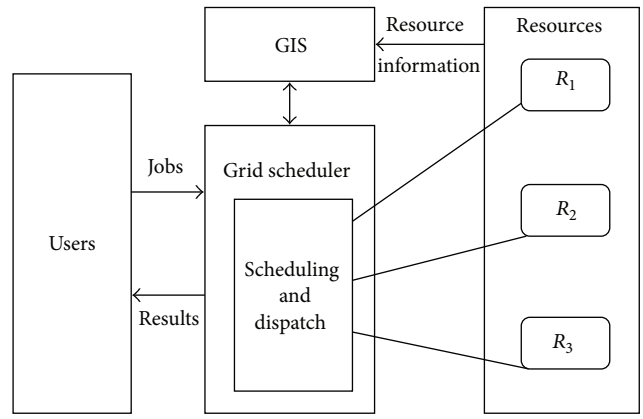

Figure 1: Proposed scheduling architecture.

(iv) computing capacity of each resource (MIPS),

(v) RT(R_j) which represents the ready time of the resource after completing the previously assigned jobs.

2.2. Proposed BSA Algorithm. A detailed description of the proposed bicriteria scheduling algorithm (BSA) is given in this section. The scheduler gets the list of tasks from the user along with the deadline (UD). The terms and abbreviations used in the algorithm are given below in Table 1.

The proposed BSA algorithm given in Algorithm 1 first calculates total completion time and then the fitness value based on failure rate. Because these are static heuristics, it is assumed that an accurate estimate of the expected execution time for each task on each machine is known prior to execution and is contained within an ETC (expected time to compute) matrix. One row of the ETC matrix contains the estimated execution times for a given task on each machine. Similarly, one column of the ETC matrix consists of the estimated execution times of a given machine for each task in the metatask. Thus, for an arbitrary task T_i and an arbitrary resource R_j, ETC(T_i, R_j) is the estimated execution time of T_i on R_j. The ETC(T_i, R_j) entry could be assumed to include the time to move the executables and data associated with task T_i from their known source to resource R_j. For cases when it is impossible to execute task T_i on resource R_j (e.g., if specialized hardware is needed), the value ETC(T_i, R_j) is set to infinity which is based on Braun et al. [25]. But in our algorithm the time to move executables and data is identified as CMT(T_i, R_j).

The ETC matrix ETC(T_i, R_j) is calculated by the formula

$$\mathrm{ETC}\left(T_i, R_j\right) = \frac{\mathrm{Length}_i}{\mathrm{Capacity}_j}, \tag{1}$$

where Length_i is the length of job in MI and $\mathrm{Capacity}_j$ is the processing capacity of resource in MIPS.

The ready time of the resource j is the earliest time in which r""esource R_j can complete the execution of all the tasks that have previously been assigned to it (based on the

Step 1: Get the list of tasks submitted by the user with its user deadline UD (T_i)
Step 2: Construct ETC (T_i, R_j) matrix of size $m \times n$ where m represents the number of tasks and n represents the number of resources involved.
Step 3: Construct RT (R_j) matrix of size $1 \times n$.
Step 4: For each task T_i in the queue and for each resource R_j where $j \in n$
do
Construct CT (T_i, R_j) matrix of size $m \times n$
Construct CMT (T_i, R_j) matrix of size $m \times n$
Construct TCT (T_i, R_j) matrix of size $m \times n$
Compute Failure rate FR (R_j) $\forall j \in n$
Create a list UDT_i for each task T_i with resources which has TCT $(T_i, R_j) \leq$ UD (T_i)
If entries in UDT_i,
select the resource with minimum FR (R_j) and Set task T_i, Resource R_j, and TCT (T_i, R_j) to CANDIDATE $(T_i, R_j, \text{TCT}(T_i, R_j))$
else
Construct Diff (T_i) = TCT (T_i, R_j) − UD (T_i)
Compute Fitness Value FV (T_i, R_j) = FFR + F_{Diff}
Set task T_i, Resource R_j with lowest fitness value and their TCT (T_i, R_j) to CANDIDATE $(T_i, R_j, \text{TCT}(T_i, R_j))$
endif
Choose task $T_{\min}$ with minimum TCT (T_i, R_j) from CANDIDATE $(T_i, R_j, TCT(T_i, R_j))$
Dispatch task $T_{\min}$ to Resource R_j and remove $T_{\min}$ from task list.
Update RT(R_j) where j is the resource to which the task $T_{\min}$ is dispatched.
Update FR (R_j), if resource R_j fails, where j is the resource to which the task $T_{\min}$ is dispatched.
done
Step 5: If there are tasks in Task_list,
Repeat step 4.
else
Compute Makespan = max $\{\text{RT}(R_j)\}$ and
Hit Rate = $T_{\text{succ}}/T_{\text{sub}}$ $\forall j \in n$
where
T_{succ} is the number of tasks successfully completed by a resource R_j without any failure and
Deadline hit = number of tasks successfully completed within deadline.
endif

Algorithm 1: Bicriteria scheduling algorithm.

ETC entries for those tasks). The ready time of the resource j is calculated by

$$\text{RT}(j) = \sum_{i=1}^{n} \text{ETC}(T_i, R_j). \tag{2}$$

The completion time for a new task T_i on resource, R_j, CT(T_i, R_j) is the machine availability time for R_j plus the execution time of task T_i on resource R_j, that is,

$$\text{CT}(T_i, R_j) = \text{ETC}(T_i, R_j) + \text{RT}(R_j). \tag{3}$$

The maximum CT(T_i, R_j) value is the metatask execution time and is called the makespan. The communication time of each task $i \in m$ on each resource $j \in n$ is given by

$$\text{CMT}(T_i, R_j) = \text{ipt}(T_i, R_j) + \text{opt}(T_i, R_j), \tag{4}$$

where the time taken for transfer of input file is calculated as the ratio of input file size IS_i to the bandwidth of the resource channel B_j

$$\text{ipt}(T_i, R_j) = \frac{\text{IS}_i}{B_j} \tag{5}$$

and the time taken for transfer of output file is calculated as the ratio of output file size OS_i to the bandwidth of the resource channel B_j

$$\text{opt}(T_i, R_j) = \frac{\text{OS}_i}{B_j}. \tag{6}$$

Total completion time of each task $i \in m$ on each resource $j \in n$ is given by

$$\text{TCT}(T_i, R_j) = \text{CT}(T_i, R_j) + \text{CMT}(T_i, R_j). \tag{7}$$

Table 1: Terms and abbreviations.

UD (T_i)	User deadline of task i in seconds
ETC (T_i, R_j)	Expected time to compute matrix of size $m \times n$ where m represents the number of tasks and n represents the number of resources involved in seconds
RT (R_j)	Ready time of resource j in seconds
CT (T_i, R_j)	Completion time of each task $i \in m$ on each resource $j \in n$ in seconds
CMT (T_i, R_j)	Communication time of each task $i \in m$ on each resource $j \in n$ in seconds
ipt (T_i, R_j)	Time taken by T_i for transfer of input files to the resource R_j in seconds
opt (T_i, R_j)	Time taken by T_i for transfer of output files to the user from the resource R_j in seconds
TCT (T_i, R_j)	Total completion time of each task $i \in m$ on each resource $j \in n$ in seconds
FR (R_j)	Failure rate of resource j
FV (T_i, R_j)	Fitness value of task i with resource j
FFR	Failure rate fitness function
F_{Diff}	Difference fitness function

This algorithm focuses on pro-active fault tolerant mechanism where failure consideration for the grid is made before the scheduling of a job and is dispatched with hopes that the job does not fail. The parameter that decides this is the failure rate FR(R_j) for all $j \in n$, which is calculated by the formula

$$\text{FR}\left(R_j\right) = \frac{T_f}{T_{\text{sub}}}, \tag{8}$$

where T_f is the number of tasks failed to be executed previously in resource j and T_{sub} is the number of tasks submitted to be executed previously in resource j.

The failure rate ranges from 0 to 1. But it is not possible to find a suitable resource with less failure rate and high capacity for all tasks. So, a difference fitness value and a failure rate fitness value are calculated, from which we can find the overall fitness value as

$$F_{\text{Diff}} = \frac{\left(\text{Diff}\left(T_i\right) - \text{Diff}_{i_{\min}}\right)}{2}. \tag{9}$$

Failure rate fitness value is calculated using the formula

$$\text{FFR} = \frac{\left(\text{FR}\left(R_j\right) - \text{FR}_{\min}\right)}{2}, \tag{10}$$

where $\text{FR}_{\min}$ is the minimum of failure rates of all the resources and $\text{Diff}_{i_{\min}}$ is the minimum difference of task i in resource $j \in n$.

Based on the overall fitness value, the jobs are allocated to the resource.

2.3. Simulation Model. The main aim of the proposed scheduling algorithm is to minimize the makespan and to improve fault tolerance of the system proactively and it is achieved by increasing the hit rate. The simulation is done with GridSim 5.0 toolkit. The architecture followed by GridSim is given in Figure 2.

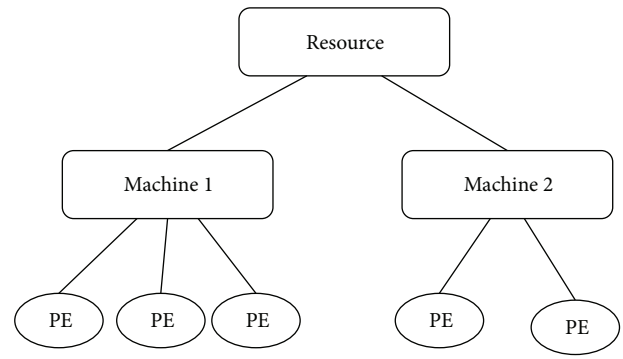

Figure 2: Gridsim Architecture.

GridSim is a very popular simulation tool for grid environment. It supports entities for simulation of a single processor and multiprocessors, heterogeneous resources that can be configured as time-shared or space-shared systems. In addition, it allows setting of the clock to different time zones to simulate the geographic distribution of resources. Furthermore, it supports entities that simulate networks used for communication among resources.

During simulation, GridSim creates a number of multithreaded entities, each of which runs in parallel to its own thread. An entity's behavior needs to be simulated within its body() method, as dictated by SimJava. A grid simulation environment needs to abstract all the entities and their time-dependent interactions in the real system. In addition, it needs to support the creation of user-defined time-dependent response functions for the interacting entities. The response function can be a function of the past, current, or both states of entities. GridSim contains entities for the users, brokers, resources, information service, statistics, and network-based, I/O.

Grid broker is the top manager of a grid environment which is responsible for maintaining the overall grid activities of scheduling and rescheduling grid resources. The grid broker gets the information of the load from grid resources and sends the Gridlet to resources for optimization scheduling. During the process, it checks the condition of the resource and schedules it if a resource leaves the grid.

GridResource is next to the grid broker in the hierarchy. It is responsible for maintaining the scheduling and load balancing of its machines. In addition, it sends an event to the 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. The GridSim resource simulator uses internal events to simulate the execution and allocation of PEs to Gridlet jobs. When a job arrives, space-shared systems start its execution immediately if there is a free PE; otherwise, it is queued. During the Gridlet assignment, the job-processing time is determined and the event is scheduled for delivery at the end of the execution time. Whenever a Gridlet finishes, an internal event is

delivered to signify the completion of the scheduled Gridlet job. The resource simulator then frees the PE allocated to it and checks if there are any other jobs waiting in the queue. If there are jobs waiting in the queue, then the resource simulator selects a suitable job depending on the policy and assigns it to the PE which is free. The selection policy can be FCFS (first-come-first-served), SJF (shortest job first), HPF (highest priority first), HRN (highest response next), and so on. If a newly arrived event happens to be an internal event whose tag number is the same as the most recently scheduled event, then it is recognized as a Gridlet completion event. If there are Gridlets in the submission queue, then depending on the allocation policy (e.g., the first Gridlet in the queue if the FCFS policy is used), GridSim selects a suitable Gridlet from the queue and assigns it to the PE or a suitable PE if more than one PE is free. After that, the completed Gridlet is sent back to its originator (broker or user) and removed from the execution set. GridSim schedules a new internal event to be delivered at the completion time of the scheduled Gridlet.

A Gridlet is an entity that contains information about the job, and its execution management details such as job length expressed in MIPS (million instructions per second), disk I/O operations, the size of input and output files, and the job originator (user). These basic parameters in GridSim determine the execution time of a job (Gridlet), the time required to transport input and output files between users and remote resources, and returning the processed Gridlets back to the originator along with the results.

A resource entity represents a Grid resource. Each resource entity may differ from the rest of the resources with respect to their different characteristics. The resource speed and the job execution time can be defined in terms of the ratings of standard benchmarks such as MIPS and SPEC (standard performance evaluation corporation). They can also be defined with respect to the standard machine. Upon obtaining the resource contact details from the Grid information service, grid brokers can query resources directly for their static and dynamic properties.

2.4. Simulation Characteristics. In this work, the Gridlets are assumed to be computationally intensive and the length of the Gridlet is considered random with a range of 50,000 to 1,00,000 MI. The Gridlets are assumed to arrive randomly following a Poisson process. The Gridlets are mutually independent and can be executed by any resource that satisfies the Gridlet requirements. Each resource can execute a single Gridlet at a time and no preemption is possible. The characteristics of resources and the scheduling parameters considered are given in Tables 2 and 3, respectively.

The definition of Gridlet, GridResource, and Machine in gridsim is as follows:

Gridlet(int gridletID, double gridletLength, long gridletFileSize, long gridletOutputSize, boolean record),

where

gridletID—the unique ID of this Gridlet,

gridletLength—the length or size (in MI) of this Gridlet to be executed in a GridResource,

Table 2: Grid resource characteristics.

No. of machines	1
No. of PE's per machine	1-2
PE ratings	5 to 50 MIPS

Table 3: Scheduling parameters and their values.

No. of gridlets	512
Gridlet length	50,000 to 1,00,000 MI
I/P file size	50 to 500 MB
O/P file size	100 to 700 MB

gridletFileSize—the file size (in byte) of this Gridlet before submitting to a GridResource,

gridletOutputSize—the file size (in byte) of this Gridlet after finish executing by a GridResource,

record—record the history of this object or not;

Machine(int Machineid, int numPE, int ratingPE),

where

id—the machine ID,

numPE—the number of PEs in this machine,

ratingPE—the rating in MIPS of a resource in this machine,

GridResource(String name, double baud_rate, long seed, ResourceCharacteristics resource, double peakLoad, double offPeakLoad, double relativeHolidayLoad, LinkedList weekends, LinkedList holidays),

where

name—the name to be associated with this entity,

baud_rate—network communication or bandwidth speed,

seed—the initial seed,

resource—an object,

peakLoad—the load during peak times,

offPeakLoad—the load during off peak times,

relativeHolidayLoad—the load during holiday times,

weekends—a linked-list contains the weekend days,

holidays—a linked-list contains the public holidays.

In this paper, two new parameters are added, that is, deadline t to Gridlet and failure rate to GridResource. Taking deadline t into account, the definition of Gridlet is extended as follows:

Gridlet(int gridletID, double gridletLength, long gridletFileSize, long gridletOutputSize, boolean record, int t).

Assume that each resource has one machine. Each machine has PEs varied from 1 to 2. The grid environment is created with 16 resources and 512 gridlets. Each PE under single machine has same processing power. Taking failure rate fr into account, the definition of GridResource is extended as follows:

GridResource(String name, double baud_rate, long seed, ResourceCharacteristics resource, double peakLoad, double offPeakLoad, double relativeHolidayLoad, LinkedList weekends, LinkedList holidays, double fr).

3. Results and Discussion

The simulation results are based on Braun et al. [25]. It is also assumed that each machine executes a single task at a time (i.e., no multitasking). The proposed BSA algorithm considers two major criterions such as user deadline of tasks and fault tolerance. The algorithm is evaluated using GridSim with 512 tasks and 16 resources.

To evaluate the heuristics for different mapping scenarios, the characteristics of the ETC matrix were varied based on several different methods given by R. Armstrong [26]. The amount of variance among the execution times of tasks in the metatask for a given machine is defined as task heterogeneity. Machine heterogeneity represents the variation that is possible among the execution times for a given task across all the machines.

An ETC matrix is said to be consistent if whenever a machine m_j executes any task t_i faster than machine m_k, then machine m_j executes all tasks faster than machine m_k. Consistent matrices were generated by sorting each row of the ETC matrix independently, with machine m_0 always being the fastest. In contrast, inconsistent matrices characterize the situation where machine m_j may be faster than machine m_k for some tasks and slower for others. These matrices are left in the unordered, random state in which they were generated. Partially consistent matrices are inconsistent matrices that include a consistent submatrix of a predefined size.

The cases discussed in this simulation are forms of consistent matrices.

(i) Low task and low machine

(ii) Low task and high machine

(iii) High task and low machine

(iv) High task and high machine.

The evaluation parameters include the hit count which is the total number of tasks successfully completed and the deadline hit which is the number of tasks successfully completed within the deadline and makespan.

The proposed algorithm is compared with the most recent scheduling algorithms FTMM [11] that depend on only fault tolerance and not user satisfaction and Min-Min algorithm which is proved as a benchmark algorithm for both static and dynamic scheduling.

3.1. Makespan. Makespan is one of the most important standard metric of grid scheduling to measure its performance. Makespan is defines as

$$\text{Makespan} = \max\left\{\text{RT}\left(R_j\right)\right\}, \quad \forall j \in n. \tag{11}$$

Makespan is used to measure the ability of grid to accommodate gridlets in less time. Figure 3 and Table 4 show the makespan comparison of the proposed BSA system and the existing systems FTMM and Min-Min.

Table 4: Comparison based on makespan (in sec).

Cases	Min-Min	FTMM	BSA
1	1345621	1300121	1284811
2	745621	565242	564011
3	2263541	2041621	1984316
4	254634	223245	220136
5	3468430	3046347	2934228

Table 5: Comparison based on hit count.

Cases	Min-Min	FTMM	BSA
1	326	368	381
2	218	280	304
3	264	298	313
4	209	239	248
5	362	394	402

3.2. Hit Count and Hit Rate. Hit count is a new metric introduced in this work. It represents the number of tasks successfully completed without considering the failure of resources.

Hit rate is the ratio of number of tasks successfully completed to the number of tasks submitted to grid and it is given by

$$\text{Hit Rate} = \frac{T_{\text{succ}}}{T_{\text{sub}}}, \quad \forall j \in n. \tag{12}$$

Figure 4 and Table 5 show the hit count comparison of the proposed BSA system and the existing FTMM and Min-Min algorithms.

3.3. Deadline Hit Count. Deadline hit count is another new metric introduced in this work. This represents the number of tasks successfully completed within user deadline. This is a measure of user satisfaction which is the key requirement in grid systems. Table 6 and Figure 5 represent the deadline hit count comparison for the three algorithms. Since the user satisfaction is the key issue addressed in this work, the average percentage of deadline hit count which decides the efficiency of proposed BSA towards it is represented in Table 7 and Figure 6.

The average percentage improvement of five different sets of 512 tasks and 16 resources based on makespan over FTMM is 1.86% and over Min-Min is 14.03%. Based on hit count, the average percentage improvement is 2.69% over FTMM and 10.5% over Min-Min.

4. Conclusion and Future Work

The proposed BSA algorithm considers both proactive fault tolerance and user deadline while scheduling the jobs. Experiments have been done for makespan that serves as

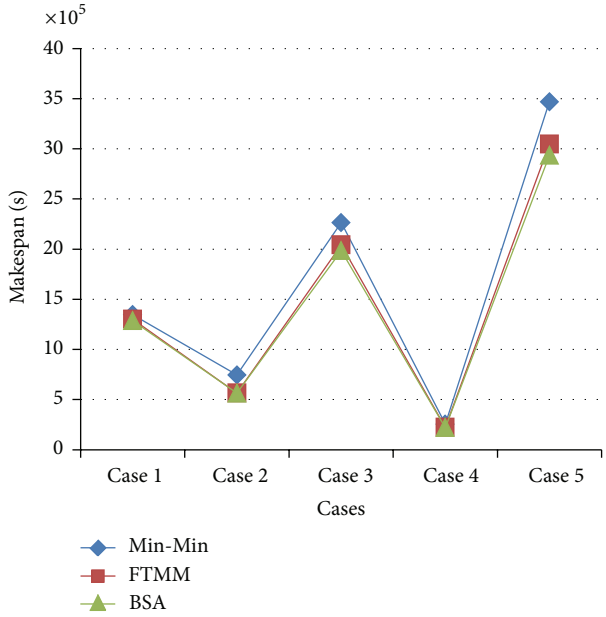

Figure 3: Comparison chart based on Makespan (in sec).

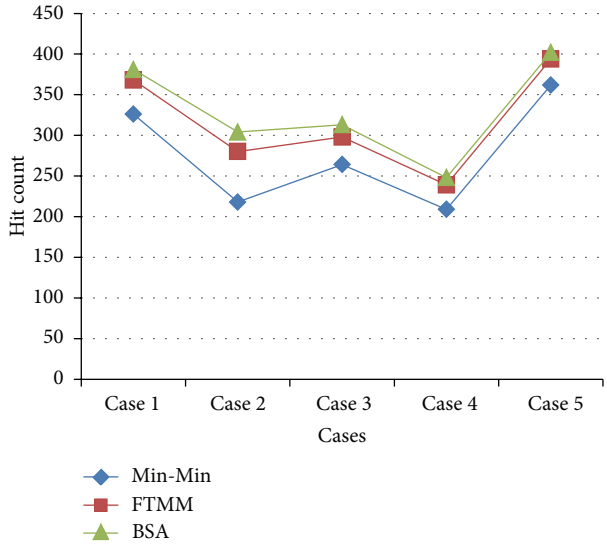

Figure 4: Comparison based on hit count.

Table 6: Comparison based on deadline hit count.

Cases	Min-Min	FTMM	BSA
1	210	242	374
2	143	189	289
3	150	211	301
4	121	199	230
5	222	321	395

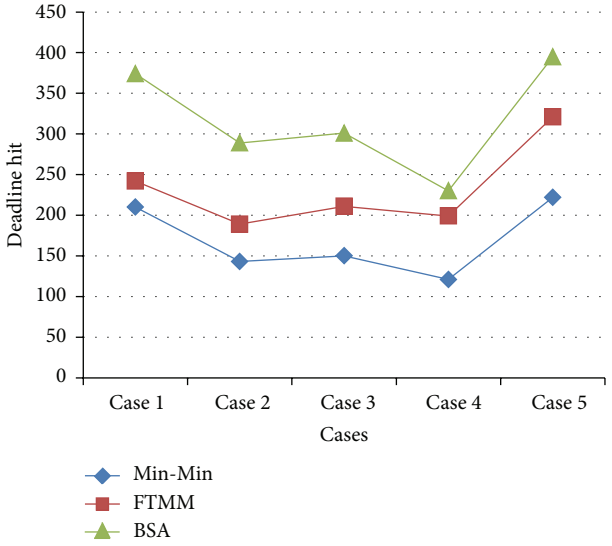

Figure 5: Comparison based on deadline hit count.

Figure 6: Average percentage of deadline hit count.

Table 7: Average percentage of deadline hit count.

	Average % of deadline count
Min-Min	61.212
FTMM	73.76
BSA	96.08

a parameter for evaluating the efficiency of the algorithm and hit count that serves as the fault tolerance parameter and deadline hit count which is a measure of user satisfaction. From Section 3 it is observed that the adoption of fault tolerance measures and consideration of user deadline of tasks achieves a better result than the existing Min-Min and FTMM algorithms. The proposed BSA algorithm considers the user deadline of each task and the failure rate of each resource at the time of scheduling which is very important in grid environment. There are many other features that should be considered while scheduling like load factor of resources,

resource availability, and so forth. So, this algorithm can be extended along with those features to achieve higher efficiency in scheduling.

References

[1] X. S. He, X. H. 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.

[2] N. Malarvizhi and V. R. Uthariaraj, "A minimum time to release job scheduling algorithm in computational grid environment," in *Proceedings of the 5th International Joint Conference on INC, IMS, and IDC*, pp. 13–18, August 2009.

[3] P. Suresh and P. Balasubramanie, "Grouping based user demand aware job scheduling approach for computational grid," *International Journal of Engineering Science and Technology*, vol. 4, no. 12, 2012.

[4] R. Buyya, M. Murshed, and D. Abramson, "A deadline and budget constrained cost-time optimization algorithm for scheduling task farming applications on global grids," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, pp. 24–27, Las Vegas, Nev, USA, 2002, http://arxiv.org/ftp/cs/papers/0203/0203020.pdf.

[5] Q. Zheng, B. Veeravalli, and C. Tham, "Fault-tolerant scheduling for differentiated classes of tasks with low replication cost in computational grids," in *Proceedings of the 16th International Symposium on High Performance Distributed Computing (HPDC '07)*, pp. 25–29, ACM, Monterey, Calif, USA, June 2007.

[6] H. Lee, D. Park, M. Hong, S. S. Yeo, S. Kim, and S. Kim, "A resource management system for fault tolerance in grid computing," in *Proceedings of the International Conference on Computational Science and Engineering (CSE '09)*, pp. 609–614, August 2009.

[7] V. Subramani, R. Kettimuthu, S. Srinivasan, and P. Sadayappan, "Distributed job scheduling on computational grids using multiple simultaneous requests," in *Proceedings of the 11th IEEE Symposium on High Performance Distributed Computing (HPDC '02)*, pp. 359–366, Edinburgh, Scotland, 2002, http://www.mcs.anl.gov/~kettimut/publications/hpdc02.pdf.

[8] 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, April 2007.

[9] 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, no. 1, 2011.

[10] A. Benoit, M. Cole, S. Gilmore, and J. Hillston, "Enhancing the effective utilisation of grid clusters by exploiting on-line performability analysis," in *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid '05)*, pp. 317–324, May 2005.

[11] 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, 2013.

[12] 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.

[13] R. Garg and A. K. Singh, "Fault tolerance in grid computing: state of the art and open issues," *International Journal of Computer Science & Engineering Survey*, vol. 2, no. 1, 2011.

[14] V. Modiri, M. Analoui, and S. Jabbehdari, "Fault tolerance in grid using ant colony optimization and directed acyclic graph," *International Journal of Grid Computing & Applications*, vol. 2, no. 1, 2011.

[15] R. Medeiros, W. Cirne, F. Brasileiro, and J. Sauve, "Faults in grids: why are they so bad and what can be done about it?" in *Proceedings of the 4th International Workshop on Grid Computing*, pp. 18–24, 2003.

[16] G. Wrzesińska, R. V. van Nieuwpoort, J. Maassen, T. Kielmann, and H. E. Bal, "Fault-tolerant scheduling of fine-grained tasks in grid environments," *International Journal of High Performance Computing Applications*, vol. 20, no. 1, pp. 103–114, 2006.

[17] H. Li, M. Muskulus, and L. Wolters, "Modeling job arrivals in a data-intensive grid," in *Proceedings of the 12th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '06)*, pp. 210–231, 2006.

[18] K. Limaye, B. Leangsuksun, Z. Greenwood et al., "Job-site level fault tolerance for cluster and grid environments," in *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER '05)*, pp. 1–9, September 2005.

[19] F. Favarim, J. da Silva Fraga, L. C. Lung, and M. Correia, "GRIDTS: a new approach for fault-tolerant scheduling in grid computing," in *Proceedings of the 6th IEEE International Symposium on Network Computing and Applications (NCA '07)*, pp. 187–194, July 2007.

[20] F. Favarim, J. da Silva Fraga, L. C. Lung, M. Correia, and J. F. Santos, "Exploiting tuple spaces to provide fault-tolerant scheduling on computational grids," in *Proceedings of the 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC '07)*, pp. 403–410, May 2007.

[21] D. Christopher, "Reliability in grid computing systems," *Concurrency and Computation: Practice and Experience*, vol. 21, no. 8, pp. 927–959, 2009.

[22] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: enabling scalable virtual organizations," *International Journal of High Performance Computing Applications*, vol. 15, no. 3, pp. 200–222, 2001.

[23] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "The physiology of the grid: an open grid services architecture for distributed systems integration," 2008, http://www.globus.org/alliance/publications/papers.php.

[24] I. Foster, J. Kishimoto, A. Savva et al., "The open Grid services architecture, version 1.5. GFD.80," Open Grid Forum, 2006.

[25] T. D. Braun, H. J. Siegel, N. Beck et al., "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810–837, 2001.

[26] R. Armstrong, *Investigation of effect of different run-time distributions on smartnet performance [M.S. thesis]*, Department of Computer Science, Naval Postgraduate School, 1997, D. Hensgen, advisor.