International Journal of Advanced Science and Technology
Vol.58, (2013), pp.13-28
http://dx.doi.org/10.14257/ijast.2013.58.02

A Hybrid Scheduling Algorithm with Load Balancing for Computational Grid

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

[1]*Research Scholar and Assistant Professor (Senior Grade), Department of CSE, Kongu Engineering College, Perundurai, Erode, Tamilnadu-638052*
[2]*Professor, Department of ECE, Kongu Engineering College, Perundurai, Erode, Tamilnadu-638052*
[1]*keerthikame@gmail.com*,[2]*kasthurisena@yahoo.com*

Abstract

Grid Computing provides seamless and scalable access to wide-area distributed resources. Since, computational grid shares, selects and aggregates wide variety of geographically distributed computing resources and presents them as a single resource for solving large scale computing applications, there is a need for a scheduling algorithm which takes into account the various requirements of grid environment. Hence, this research proposes a new scheduling algorithm for computational grids that considers load balancing, fault tolerance and user satisfaction based on the grid architecture, resource heterogeneity, resource availability and job characteristics such as user deadline. This algorithm reduces the makespan of the schedule along with user satisfaction and balanced load. A simulation is conducted using Grid Simulator Toolkit (GridSim). The simulation results shows that the proposed algorithm has better makespan, hit rate and resource utilization.

Keywords: *Load Balancing, User deadline, Fault tolerance, Scheduling, Grid Computing, Resource Utilization*

1. Introduction

Grid Computing is an important paradigm that supports sharing and access to a large amount of computational and storage resources which are heterogeneous and geographically distributed. The two variations of grid enclose computational grids and data grids. Computational grid exploits the synergy between a set of interconnected grid nodes to reach a common goal to solve massive computational problems [11, 35]. Data intensive computing is concerned with addressing the technical challenges generated by the ever growing demands for processing large scale data sets [27]. The main aspect of grid computing that is to be considered is the dynamicity of resources. The resources of grid can be freely added or withdrawn at any time according to the owner's discretion. So, the performance of grid nodes and their load frequently changes with respect to time [1]. In Grid environment, scheduling is an important aspect to be taken care since it is known to be a NP hard problem [5]. Grid Scheduling is a process of splitting a larger problem to a number of sub problems and allocating those tasks to the resources based on resource capability and job requirements. The Scheduler plays an important role in computational grids. The selection of proper scheduling algorithm should be of at most care in order to maximize the throughput.

Since scheduling is a NP hard problem, finding an exact solution for larger problems is not possible. Instead an approximate solution can be achieved. Many heuristic scheduling algorithms have been proposed that provide approximate solutions to problems. [25, 36]

ISSN: 2005-4238 IJAST

presents many heuristic methods for static and dynamic scheduling. But those algorithms lacks of balanced load among geographically distributed heterogeneous grid resources. The main objective of heuristic algorithms is to minimize makespan which is the maximum completion time spent for execution of a batch of tasks [33]. Among most of the static scheduling algorithms, Genetic algorithm (GA) has better makespan [32] and in case of dynamic scheduling, min-min algorithm has better makespan and they are considered as the benchmark algorithms [24,37]. One of the aspects considered in this research is load balancing. They attempt to improve the response time and also ensure maximum utilization of resources available. Load balancing can be defined by the following policies [16].

1. Information Policy - specifies what workload information is to be collected, when it is to collected and from where.
2. Triggering Policy - determines the appropriate at which to start a load balancing operation.
3. Resource type Policy - classifies resource as a server or a receiver of tasks according to its availability status.
4. Location Policy – uses the results of the resource type policy to find a suitable partner for a resource provides or a resource receiver.
5. Selection Policy – defines the tasks that should be migrated from overloaded resource to the idlest resources.

Load balancing algorithms can be classified into two categories such as static algorithms and dynamic algorithms. In static load balancing, decision is made at compile time when resource requirements are estimated. Dynamic load balancing algorithm allocates/reallocates resources at runtime. The second aspect considered in this research is fault tolerance which deals with failure of resources dynamically. When a task is submitted to a grid broker, it schedules the task to resources available based on the factors and allocates task to the selected resource. If that resource fails due to any of the following reasons, that should also be handled by the proposed algorithm. Failures may be network failure, resource failure etc. There are two types of failure handling mechanisms - Proactive and Passive. In pro-active mechanisms, the failure consideration for the grid is made before the scheduling of a job, and dispatched with hopes that the job does not fail. Whereas, post-active mechanisms handles the job failures after it has occurred. However, in the dynamic systems only post-active mechanism is relevant [23]. The third aspect is user satisfaction which is achieved by considering the user deadline of the tasks submitted. The tasks are received along with its deadline. The task is expected to be executed within the deadline.

2. Related Works

This section discusses some of the recent algorithms designed for load balancing, fault tolerance and user satisfaction based scheduling algorithms. An adaptive scheduling algorithm that considers both quality of service (QoS) and non-dedicated computing is proposed in [13]. Similar to existing task scheduling algorithms, this scheduling algorithm is designed to achieve high throughput computing. A minimum time to release job scheduling algorithm is proposed [22] 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. A Grouping based Scheduling algorithm is proposed in [31] in which user deadline and reduces communication overhead by adopting the grouping technique.

A cost optimization scheduling algorithm to optimize the cost to execute the jobs is described in [3]. 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. A fault-tolerant scheduling for differentiated classes of independent tasks is introduced in [38]. 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. A fault tolerance based resource management service which considers different types of failure and QoS requirement is proposed in [19]. This algorithm fails to concentrate on user satisfaction and execution time. A greedy meta-scheduling algorithm based on multiple simultaneous requests is proposed in [28]. 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.

An application demand aware algorithm [15] considers application demand of the jobs for scheduling. It produces better user satisfaction and fault rate of the resources are not considered. A Prioritized user demand algorithm is proposed [29] 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 novel method of modeling job execution on grid compute clusters is proposed in [5]. This algorithm uses Performance Evaluation Process Algebra (PEPA) as the system description formalism, capturing both workload and computing fabric.

In our previous work [17], 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 [30]. This strategy does not consider the occurrence of resource failure. The fault tolerant algorithm discussed by [26] 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 additional 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. A DAG mechanism [34] to enter tasks and thereby brings out an efficient algorithm namely Ant Colony Optimization algorithm.

In [23], fault tolerance in grid environment can be divided into pro-active and post-active mechanisms. The pro-active mechanisms consider the job failure history of each resource before scheduling of a job and dispatches jobs to resources with hopes that the job does not fail. Whereas, post-active mechanisms handles the job failures after it has occurred. However, for dynamic grid systems only post-active mechanism is relevant than the pro-active mechanisms. For Grid environment, [12] proposed that there are several reasons for workflow execution failure. The reasons are variation in the execution environment configuration, non-availability 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 [20, 21]. Since, the Grid infrastructure is still emerging, there are only a few works that has been done for job failure analysis and they all find it tough to collect traces at hierarchical levels. A new fault tolerance approach [6] with several masters called brokers. The Grid broker receives jobs from their users and divides 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. In [7], all communication between brokers/masters and resources/workers is done exclusively through the tuple space.

The survey done by [4] 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 [8-10]. 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 grown within industry and science. A hybrid scheduling policy with fault tolerance and load balancing has been introduced by [14]. In the first phase, a static load balancing policy selects the desired sites. If any site is unable to complete the assigned job, a new site will be located using the dynamic load balancing policy.

In our previous work [18], we have proposed a new Bicriteria scheduling algorithm that considers both user satisfaction and fault tolerance. The pro-active 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. The main objective of this paper is to develop a scheduling algorithm which reduces makespan and improves resource utilization. It also ensures that the tasks are completed within the user expected deadline. While scheduling the jobs, failure rate of the resources and the load of each resource is also considered. The proposed algorithm improves system performance, user satisfaction, resource utilization 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. The materials and methods section describes about 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, FTMM algorithm and BSA algorithm in results and discussion section.

3. Materials and Methods

3.1. 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 Multi Criteria scheduling

algorithm (MCSA) works. The Grid Information Service (GIS) holds the information of all the resources such as resource id, resource availability and 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 load of resource, its failure rate, resource capacity and user deadline.

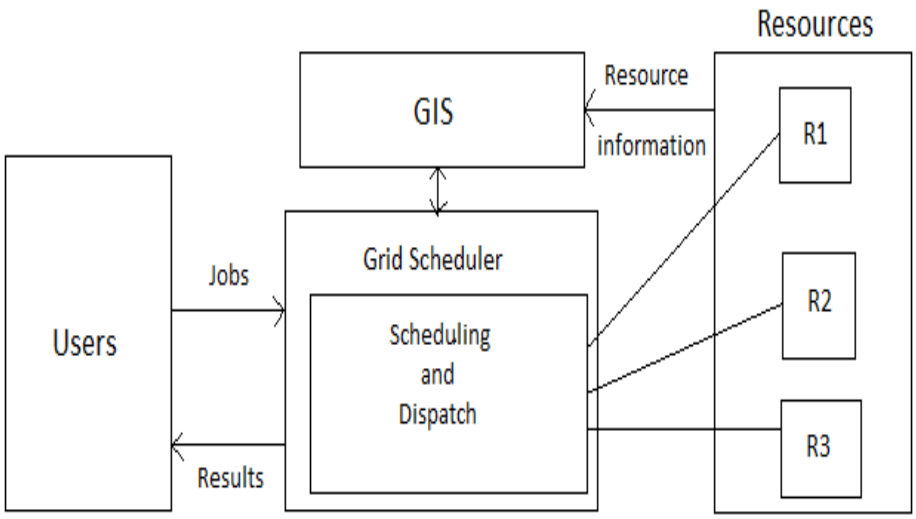

Figure 1. Proposed Scheduling Architecture

Each resource differs from other resources in many ways that includes number of processing elements, processing speed, internal scheduling policy and its load factor etc. Similarly each job differs from other jobs by execution time, deadline, time zone etc. The static mapping of meta tasks is done in which each machine executes one task at a time. It is assumed that the size of the meta tasks, number of resources, expected execution time of each task in each machine are known priori. The proposed scheduling architecture follows the hierarchy in Figure 2. Machine is a collection of Processing Elements (PE) and it acts as a processing entity manager. In Gridsim simulator, the resource module simulates and allocates gridlets to available PE's. The gridlets received are queued and then assigned to PE's based on resource availability time and gridlet's expected execution time. When a gridlet is executed, then an internal event is delivered to machine regarding the completion of gridlet. Then the PE is freed and made available for other gridlets in queue. The selection policy is where scheduling is to be done. This research determines the scheduling policy based on user deadline, load of PE's and failure rate of PE's.

The resources are categorized as overloaded, under loaded and normally loaded based on threshold value calculated. 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 [8] wherein the scheduling problem is defined by

- A number of independent tasks to be allocated to the available grid resources.
- Number of resources is available to participate in the allocation of tasks.
- Workload of each task (MI).
- Computing capacity of each resource (MIPS)

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

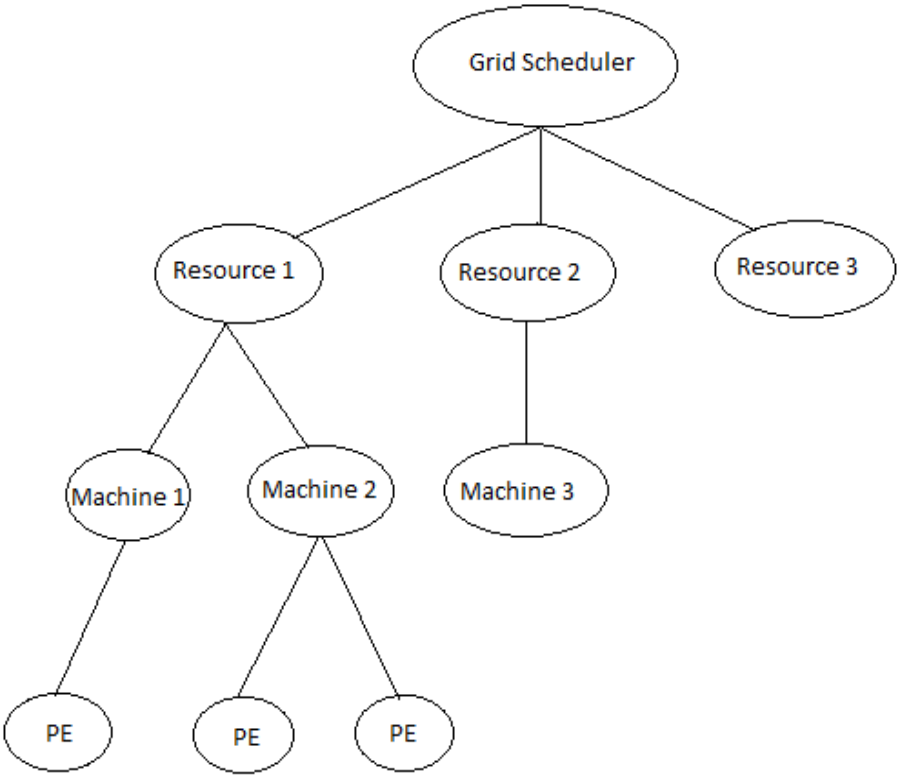

Figure 2. Scheduling Hierarchy

3.2. Proposed MCSA Algorithm

The proposed MCSA algorithm works as follows. The gridlets are submitted to the grid broker or scheduler. The Grid Information Service (GIS) maintains the information about all resources. The grid scheduler gets information about the resources like load of a resource, its availability time and capacity from GIS. Grid Scheduler receives the gridlets with user deadline $UD(T_i)$. The gridlet's information such as its length in Million Instructions (MI), is used to calculate the execution time $ETC(T_i, R_j)$of each gridlet in each of the available resources by using the formula,

$$ETC(T_i, R_j) = \frac{Length_i}{Capacity_j} \quad (1)$$

where

$Length_i$ is the length of job in MI and
$Capacity_j$ is the processing capacity of resource in MIPS.

With the ready time information $RT(R_j)$ available for each resource at GIS, the algorithm calculates the completion time using the formula,

$$CT(T_i, R_j) = ETC(T_i, R_j) + RT(R_j) \quad (2)$$

Information available about each resource is the communication time $CMT(Ti, Rj)$ is used to calculate the total completion time of each gridlet at each resource as

$$TCT(T_i, R_j) = CT(T_i, R_j) + CMT(T_i, R_j) \quad (3)$$

The failure information of resources such as number of gridlets submitted to a resource T_{sub} and number of gridlets successfully completed $T_{succ}$and number of gridlets not completed successfully T_f is also available in GIS which helps in calculating the failure rate as

$$FR(R_j) = \frac{T_f}{T_{sub}} \quad (4)$$

Since the main focus is on load balancing, calculation of load of each PE, Machine and Resource becomes essential. Load of each PE is calculated by using the weighted sum of squares method as follows.

$$Load(PE_i) = \sqrt{\sum_{k=1}^{n}(a_k {L_k}^2)} \quad (5)$$

where L_k is the load attribute considered in our algorithm. The load attribute considered in our algorithm is the CPU utilization in seconds. Hence the load of PE is given by,

$$Load(PE_i) = \frac{\sum_{j=0}^{n} MI_i}{MIPS_i \times AT_i} \quad (6)$$

where n is the number of tasks allocated to PEi. Machine is the collection of PE's and the average load of each machine Mi is calculated using the formula,

$$AL\ (M_i) = \frac{\sum_{k=1}^{n} Load\ (PE_k)}{n} \quad (7)$$

where n is the number of PE's under Machine i. Resource is the collection of machines and the average load of each resource Ri is calculated by,

$$AL\ (R_i) = \frac{\sum_{k=1}^{n} AL\ (M_k)}{n} \quad (8)$$

where n is the number of machines under resource i. The average load of the system is calculated as,

$$AL = \frac{\sum_{k=1}^{n} AL\ (R_k)}{n} \quad (9)$$

where n is the number of resources in the system. In order to balance the load of the resources in grid, a balance threshold Ω is defined such that the average loads of each resource $AL(Ri)$ to be less than the balance threshold of the system.

$$\Omega = AL + \sigma \quad (10)$$

where σ is the standard deviation of the load of the system which is defined as below:

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

where N is the number of resources in the system. The New load of PE can be calculated by,

$$Newload(PE_j) = Load(PE_j) + \frac{MI_i}{MIPS_j + AT_j} \quad (12)$$

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.

3.3. Simulation Setup

The main aim of the proposed scheduling algorithm is to minimize the makespan and to improve fault tolerance of the system proactively with balanced load of resources. Fault tolerance is achieved by increasing the Hit rate. User Satisfaction is achieved by improving the deadline hit count. The simulation is done with Gridsim 5.0 toolkit.

Number of Resources : 16

Number of Tasks : 512

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 pre-emption is possible. The characteristics of resources and the scheduling parameters considered are given in Table 1 and Table 2 respectively.

Table 1. Grid Resource Characteristics

No. of machines	1-4
No. of PE's per machine	1-2
PE ratings	5 to 50 MIPS

Table 2. 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

Step 1: Get the Task_list T of tasks submitted by the user with its user deadline $UD\ (T_i)$

Step 2: Get the list R of resources available in grid from GIS and initialize the counters Deadline Hit Count and Hit Count

Step 3: Construct $ETC(T_i, R_j)$ matrix of size m × n where m represents the number of tasks and n represents the number of resources involved.

Step 4: For all resources R_j in R, where $1 \leq j \leq n$, and n denotes number of resources,
do

4.1: Calculate Failure rate

4.2: Calculate Ready Time

$$RT(R_j) = \sum_{i=1}^{n} ETC(T_i, R_j)$$

where n is the number of tasks submitted to R_j.

4.3: Calculate Load of each Processing Element, Average Load of each machine and Average Load of each resource

done

Step 5: Calculate Average Load of the system and Balance Threshold

Step 6: Create a list of underloaded resources UR which has $AL(R_j) < \Omega$.

Step 7: For each task in T_i in queue and for each resource R_j,
do

7.1: Construct (T_i, R_j) , $CMT(T_i, R_j)$, $TCT(T_i, R_j)$ matrix of size m×n

done

Step 8: For all task T_i in Task_list T,
do

8.1: Create lists UT_{i_1} and UT_{i_2} with resources that has $TCT(T_i, R_j) \leq UD\ (T_i)$ and $TCT(T_i, R_j) > UD\ (T_i)$ respectively.

8.2: Sort the lists 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 check for load balancing using the equation (12) and if the load is balanced, dispatch T_i to resource R_j and Increment Deadline Hit Count and Hit Count after receiving the completion status. Otherwise, select next resource and repeat step 8.4.

else if entries in ULT_{i_2},

Select the first resource in the list for task T_i and check for load balancing using the equation (12) and if the load is balanced, dispatch T_i to resource $R_j$and Increment Hit Count after receiving the completion status. Otherwise, select next resource and repeat step 8.4.

8.5: Remove task T_i from Task_list T.

8.6: Update $RT(R_j)$and $FR(R_j)$ where j is the resource to which the task T_i is dispatched.

done

Step 9: If there are tasks in Task_list T, Repeat steps from 4.3.

else

Compute $Makespan = max\{RT(R_j)\}$ and $Hit\ Rate = \frac{T_{succ}}{T_{sub}}\ \forall\ j \in n$

where T_{succ} is the number of tasks successfully completed by a resource R_j without any failure and T_{sub} is the number of tasks failed to be executed by a resource R_j.

Compute Resource Utilization

$$RU(R_j) = \frac{\sum_{j=0}^{n} MI_i}{MIPS_j \times AT_j} \times 100$$

Compute Average Resource Utilization

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

endif

Algorithm 1. MCSA Algorithm

4. Results and Discussion

In this section, the simulation results of the proposed algorithm are compared with the some of the existing algorithms which are recent in scheduling with fault tolerance and satisfying user satisfaction. The existing algorithm considered for comparison are FTMM and BSA which are discussed in our previous work [17, 18] and the benchmark scheduling algorithm Min-Min discussed in [8]. The proposed algorithm differs from these existing algorithms in such a way that these algorithms do not deal with load balancing but the proposed MCSA takes load balancing as an important factor along with those parameters considered in the existing algorithms. The cases discussed in this simulation are forms of consistent matrices.

- Low task and Low Machine
- Low task and High Machine
- High Task and Low Machine
- High Task and High Machine

The proposed MCSA algorithm is evaluated for makespan, hit rate, deadline hit count and resource utilization and compared with the other three algorithms. Table 3 and Figure 3 represent the makespan comparison of the proposed MCSA and the existing Min-Min, FTMM and BSA algorithms. The analysis show that the makespan is minimized to some extent in most of the cases and there is a possibility of increased makespan in few cases.

Table 3. Comparison based on Makespan (in sec)

CASES	MIN-MIN	FTMM	BSA	MCSA
1	1245621	1130121	1000811	987600
2	746543	626549	554387	589070
3	1963786	1842149	1687876	1484316
4	559650	453986	342111	287605

Table 4. Comparison based on Hit Count

CASES	MIN-MIN	FTMM	BSA	MCSA
1	311	332	381	381
2	233	278	297	309
3	265	287	312	324
4	341	356	362	358

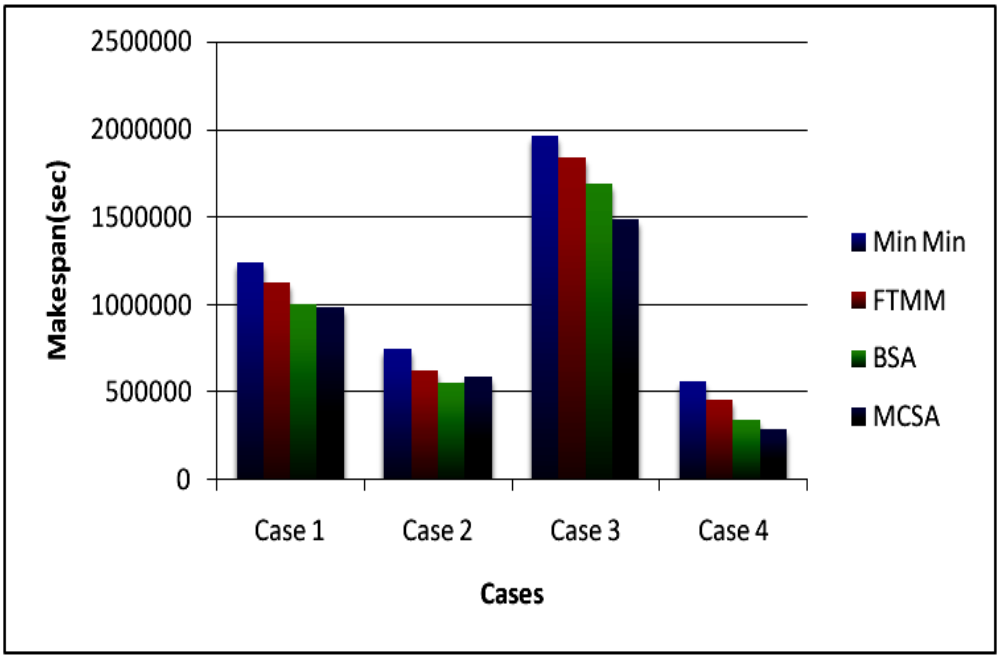

Figure 3. Comparison based on Makespan (sec)

Figure 4 and Table 4 show the hit count comparison of the proposed MCSA algorithm and the existing BSA, FTMM and Min-Min algorithm. It is inferred that the number of tasks completed (hit count) in MCSA algorithm is more or less similar to BSA algorithm but has a notable deviation with the other two algorithms FTMM and Min-Min.

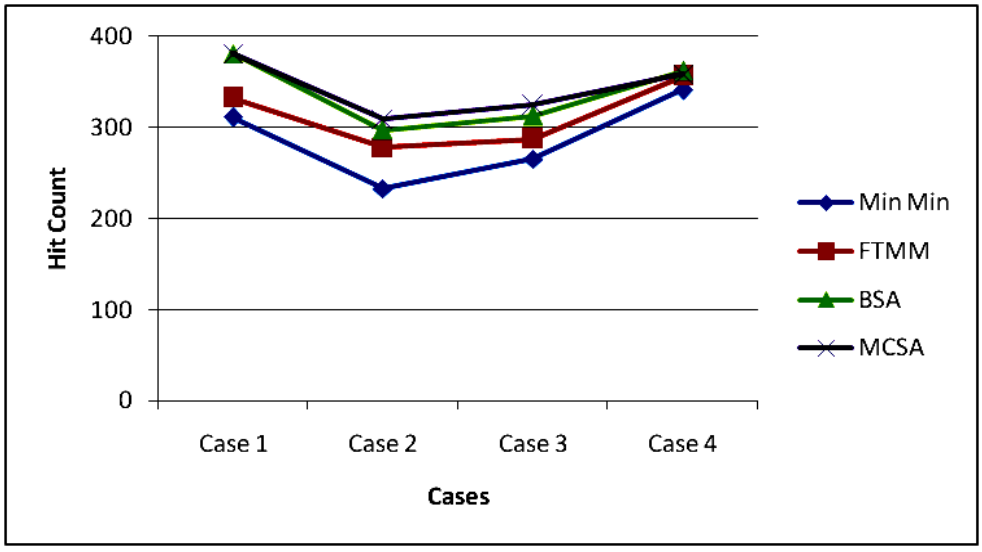

Figure 4. Comparison based on Hit Count

Figure 5 and Table 5 show the analysis of proposed MCSA with the existing BSA, FTMM and Min-Min based on deadline hit count, the evaluation parameter considered to evaluate on the basis of user satisfaction. When the deadline hit count is more, it is inferred

that the user satisfaction is more. The analysis shows that the user satisfaction is more with the MCSA algorithm than the other three algorithms.

Table 5. Comparison based on Deadline Hit Count

CASES	MIN-MIN	FTMM	BSA	MCSA
1	213	238	371	372
2	148	197	283	296
3	170	226	292	302
4	231	245	296	290

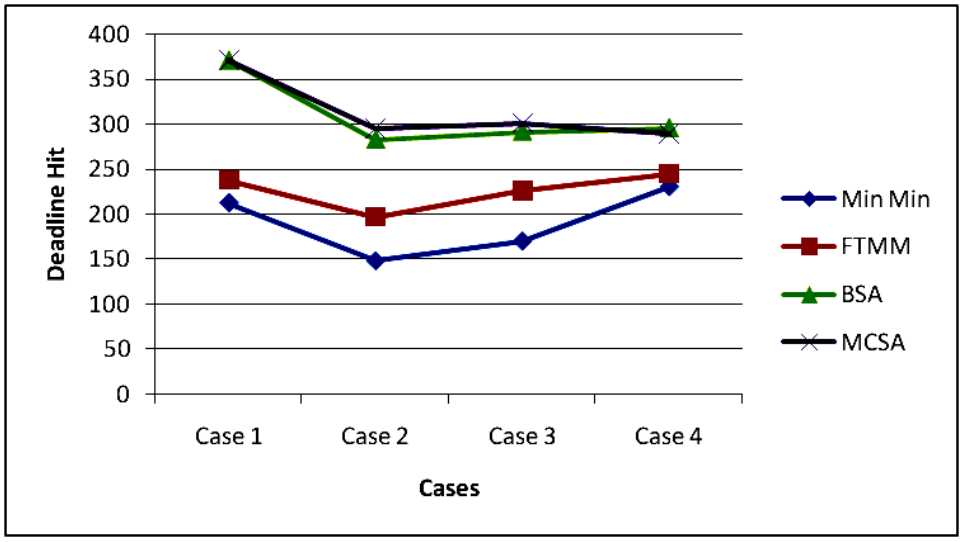

Figure 5. Comparison based on Deadline Hit Count

An important parameter used in this work to analyse the load balancing strategy of the proposed algorithm is the average resource utilization and is expressed in percentage.

Table 6. Comparison based on Average Resource Utilization

CASES	MIN-MIN	FTMM	BSA	MCSA
1	70	77	78	90
2	68	74	75	89
3	73	80	83	92
4	69	74	78	84

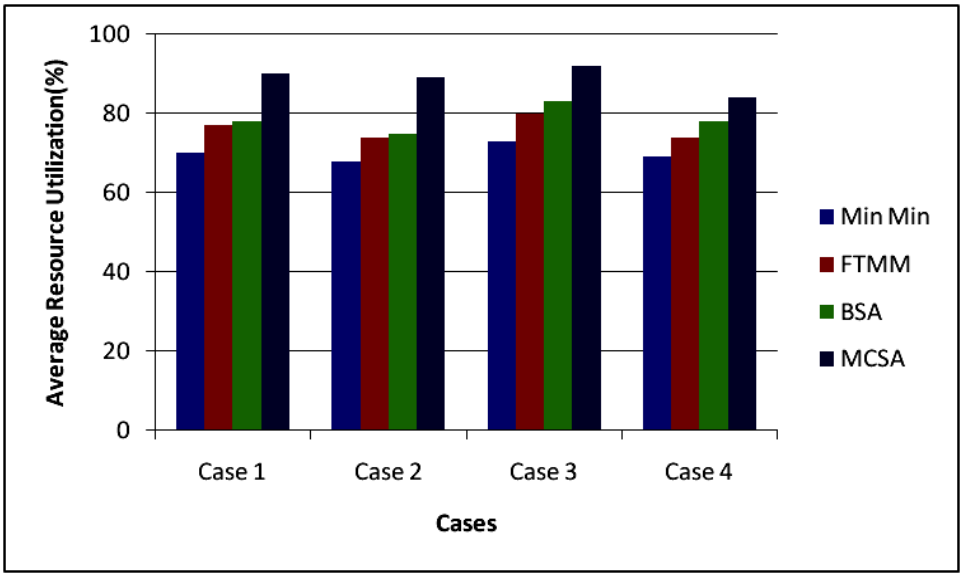

Figure 6. Comparison based on Average Resource Utilization

Table 6 and Figure 6 illustrates the average resource utilization of the proposed MCSA is relatively high when compared to the existing algorithms. The average percentage improvement of four different sets of 512 tasks and 16 resources based on makespan over BSA is 5.7%and FTMM is 18.7% and over Min-Min is 28.7%. Based on hit count, the average percentage improvement is 0.9% over BSA, 5.8% over FTMM and 10.8% over Min-Min.

5. Conclusion and Future Work

The proposed MCSA Algorithm implements proactive fault tolerance, user deadline and load balancing for scheduling the jobs. Experiments have been done for makespan that serves as 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 and finally average resource utilization that serves as the evaluation parameter for proper load balancing. From the results and discussion section it is observed that the adoption of fault tolerance measures and consideration of user deadline of tasks and balance threshold achieves a better result than the existing min-min, FTMM and BSA algorithms. The proposed MCSA algorithm considers the user deadline of each tasks and the failure rate, load of each resource at the time of scheduling which are very important in grid environment. This can be extended in future with factors for reducing the communication overhead of the grid system.

References

[1] A. Torkestani, " A new approach to the job scheduling problem in computational grids", Cluster Computing, DOI 10.1007/s10586-011-0192-5, vol. 15, no. 3, **(2012)**, pp. 201-210.
[2] B. Anne, C. Murray, G. Stephen and H. Jane, "Enhancing the effective utilization of Grid clusters by exploiting on-line performability analysis", IEEE International symposium on Cluster Computing and the Grid (CCGRID), http://dl.acm.org/citation.cfm?id=1169483, **(2005)**, pp. 317-324.
[3] R. Buyya, M. Murshed and D. Abramson, "A deadline and budget constrained cost-time optimization algorithm for Scheduling task farming applications on global grids", Proceedings of the international conference on parallel and distributed processing techniques and applications, Las Vegas, USA, pp. 24-27. http://arxiv.org/ftp/cs/papers/0203/0203020.pdf, **(2002)**.

[4] D. Christopher, "Reliability in grid computing systems", Concurrency and Computation: Practice and Experience, John Wiley & Sons, Ltd, DOI: 10.1002/cpe.1410, **(2009)**.
[5] A. Di Stefano and G. Morana, " A bio-inspired distributed algorithm to improve scheduling performance of multi-broker grids", Springer Natural Computing, DOI 10.1007/s11047-012-9319-8, vol. 11, no. 4, **(2012)**, pp. 687-700.
[6] F. Favarim, D. Joni Silva Fraga, L. C. Lung and M. Correia, " GRIDTS: A New Approach for Fault-Tolerant Scheduling in Grid Computing", Proceedings of Sixth IEEE International Symposium on Network Computing and Applications, **(2007)**, pp. 187-194.
[7] F. Favarim, D. Joni Silva Fraga, L. C. Lung, M. Correia, and J. F. Santos, "Exploiting tuple spaces to provide fault-tolerant scheduling on computational grids", 10th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing, **(2007)**, pp. 403-411.
[8] 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. 2, **(2001)**, pp. 200-222.
[9] I. Foster, C. Kesselman, J. Nick and S. Tuecke, "The physiology of the Grid: An open Grid services architecture for distributed systems integration", http://www.globus.org/alliance/publications/papers.php, **(2008)**.
[10] I. Foster, J. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell and J. Von Reich, " The open Grid services architecture", version 1.5. GFD.80, Open Grid Forum, **(2006)**.
[11] P. Ghosh and S. K. Das, "Mobility-aware cost-efficient job scheduling for single-class Grid jobs in a generic mobile Grid architecture", Future Gener. Comput. Syst., vol. 26, **(2010)**, pp. 1356-1367.
[12] G. Wrzesinska, R. V. van Nieuwport, J. Maassen, T. Kielmann and H. E. Bal, " Fault-tolerance scheduling of fine grained tasks in Grid environment", International Journal of High Performance Applications, vol. 20, no. 1, **(2006)**, pp. 103-114.
[13] H. Xiaoshan, X.-H. Sun and G. Von Laszewski, "QoS Guided Min-min Heuristic for Grid Task Scheduling", Journal of Computer Science and Technology, **(2003)**, pp. 442-451.
[14] J. Balasangameshwara and N. Raju, " 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] J. Lin, B. Gong, H. Liu, C. Yang and Y. Tian, "An Application Demand aware Scheduling Algorithm in Heterogeneous Environment", IEEE proceedings of the 11th International Conference on Computer Supported Cooperative Work in Design, IEEE Xplore press, Melbourne, Vic, **(2007)**, pp. 509-604.
[16] H. D. Karatza, "Job scheduling in heterogeneous distributed systems", Journal of Systems and Software, **(1994)**, pp. 203-212.
[17] 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, doi:10.3844/ ajassp.2012.2046.2051, **(2013)**.
[18] P. Keerthika and N. Kasthuri, "An Efficient Grid Scheduling Algorithm with Fault Tolerance and User Satisfaction", Mathematical Problems in Engineering, Mathematical Problems in Engineering, vol. 2013, Article ID 340294, http://dx.doi.org/10.1155/2013/340294, **(2013)**.
[19] H. Lee, D. Park, M. Hong, S.-S. Yeo, S. K. Kim and S. H Kim, " A Resource Management System for Fault Tolerance in Grid Computing", IEEE International Conference on Computational Science and Engineering, DOI 10.1109/CSE.2009.257, **(2009)**.
[20] H. Li, M. Muskulus and L. Wolters, "Modeling job arrivals in a data-intensive grid", proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP), **(2006)**, pp. 210-23.
[21] M. Limaye, B. Leangsuksun, Z. Greenwood, S. L. Scott, C. Engelmann, R. Libby and K. Chanchio, "Job-site level fault tolerance for cluster and grid environments", Proceedings of IEEE Cluster, **(2005)**, pp. 1-9.
[22] N. Malarvizhi and V. Rhymend Uthariaraj, "A Minimum Time to Release Job Scheduling Algorithm in Computational Grid Environment", IEEE Fifth International Joint Conference on INC, IMS, IDC.http://doi.ieeecomputersociety.org/ 10.1109/NCM.2009.373, **(2009)**.
[23] R. Medeiros, W. Cirne, M. Brasileiro and J. Sauve, "Faults in grids: why are they so bad and what can be done about it?", Proceedings of the 4th international workshop, **(2003)**, pp. 18-24.
[24] M. Maheswaran, S. Ali, H. Jay Siegel, D. Hensgen, R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems", J. Parallel Distrib. Comput. vol. 59, **(1999)**, pp. 107-131.
[25] N. Fujimoto and K. Hagihara, "A comparison among grid scheduling algorithms for independent coarse-grained tasks", Proceedings of the 2004, International Symposium on Applications and the Internet Workshops (SAINTW'04), vol. 26-30, **(2004)**, pp. 674-680.
[26] R. Garg and A. Kumar Singh, "Fault Tolerance in grid computing: state of the art and open issues", International Journal of Computer Science & Engineering Survey (IJCSES), vol. 2, no. 1,

DOI:10.5121/ijcses.2011.2107, **(2011)**.
[27] C. Rosas, A. Sikora, E. Cesar, J. Jorba and A. Moreno, "Improving Performance on Data-intensive Applications Using a Load Balancing Methodology Based on Divisible Load Theory", International Journal of Parallel Programming, DOI: 10.1007/s10766-012-0199-4, **(2012)**.
[28] V. Subramani, R. Kettimuthu, S. Srinivasan and P. Sadayappan, "Distributed Job Scheduling on Computational Grids Using Multiple Simultaneous Requests", Proceedings of the 11th IEEE Symposium on HPDC 2002), Edinburgh, Scotland, "www.mcs.anl.gov/~kettimut/ publications/ hpdc02.pdf", pp. 359-366, **(2002)**.
[29] P. Suresh, P. Balasubramanie and P. Keerthika, "Prioritized User Demand Approach for Scheduling Meta Tasks on Heterogeneous Grid Environment", International Journal of Computer Applications (0975-8887), vol. 23, no. 1, **(2011)**.
[30] 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, http://www.europeanjournalofscientificresearch.com/ ISSUES/EJSR_74_4_14.pdf, **(2012)**.
[31] P. Sures 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, http://www.ijest.info/docs/ IJEST12-04-12-093.pdf, **(2012)**.
[32] T. D. Braun, H. Jay Siegel, N. Beck, L. L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen and R. F. Freund, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems", Parallel Distrib. Comput., vol. 61, **(2001)**, pp. 810-837.
[33] L.Y. Tseng, Y. H. Chin and S. C. Wang, "The anatomy study of high performance task scheduling algorithm for Grid computing system", Journal of Computer Standards & Interfaces, DOI: 10.1016/j.csi.2008.09.017, vol. 31, no. 4, **(2010)**, pp. 713-722.
[34] 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, DOI:10.5121/ijgca.2011.2102, vol. 2, no. 1, **(2011)**.
[35] J. Wu, X. Xu, P. Zhang, C. Liu, "A novel multi-agent reinforcement learning approach for job scheduling in Grid computing". Future Gener, Comput. Syst. 27, pp. 430-439, **(2011)**.
[36] F. Xhafa and A. Abraham, "Computational models and heuristic methods for Grid scheduling problems", Future Gener, Comput. Syst., vol. 26, **(2010)**, pp. 608-621.
[37] Y. Han, C. Jiang, Y. Fu and X. Luo, "Resource scheduling algorithms for grid computing and its modeling and analysis using Petri Net", GCC (2), LNCS 3033, Springer, Shanghai, China, **(2004)**, pp. 73-80.
[38] Q. Zheng, B. Veeravalli and C. Tham, "Fault-tolerant Scheduling for Differentiated Classes of Tasks with Low Replication Cost in Computational Grids, ACM, HPDC'07, DOI:10.1145/1272366.1272409, **(2007)**.

