International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:9, No:2, 2015

A Budget and Deadline Constrained Fault Tolerant Load Balanced Scheduling Algorithm for Computational Grids

P. Keerthika, P. Suresh

***Abstract*—**Grid is an environment with millions of resources which are dynamic and heterogeneous in nature. A computational grid is one in which the resources are computing nodes and is meant for applications that involves larger computations. A scheduling algorithm is said to be efficient if and only if it performs better resource allocation even in case of resource failure. Resource allocation is a tedious issue since it has to consider several requirements such as system load, processing cost and time, user's deadline and resource failure. This work attempts in designing a resource allocation algorithm which is cost-effective and also targets at load balancing, fault tolerance and user satisfaction by considering the above requirements. The proposed Budget Constrained Load Balancing Fault Tolerant algorithm with user satisfaction (BLBFT) reduces the schedule makespan, schedule cost and task failure rate and improves resource utilization. Evaluation of the proposed BLBFT algorithm is done using Gridsim toolkit and the results are compared with the algorithms which separately concentrates on all these factors. The comparison results ensure that the proposed algorithm works better than its counterparts.

***Keywords*—**Grid Scheduling, Load Balancing, fault tolerance, makespan, cost, resource utilization.

I. Introduction and Related Works

THE computational power of individual computers is rapidly increasing from time to time. For problem solving in the fields like earth system sciences, financial modeling and high energy physics, the approaches involving computation are widely used. But for these applications, the computational power of a single computer is not sufficient. It has limited resources and is not suitable for computation-intensive applications. In order to meet the computational demand, powerful distributed and parallel systems with more number of processors are developed. But few applications like parameter search problems need more number of resources which led to a solution of collecting and utilizing distributed resources owned by different institutions and domains. This distributed computing infrastructure is called grid.

Based on functionality, grid can be classified as computational grid and data grid. The resources involved in computational grids are computational resources such as processors. It is mainly used for computation intensive applications and data intensive applications. The application which requires more time for computation are termed as computation intensive applications and the applications which requires more time for data retrieval than computation are termed as data intensive applications. In data grid, the resources are storage resources like memory and mainly deal with data storage.

A grid system comprises of a scheduler, grid portal and a Grid Information Service (GIS). The scheduler or the grid broker is responsible for mapping of tasks to its suitable resources. It allows the users to request for resource allocation. This process is termed as scheduling. Scheduling can be varied as static scheduling and dynamic scheduling. The Users communicates with the scheduler through grid portal. They have several Quality of Service (QoS) requirements of their task towards execution. The QoS requirements can be based on processing power, operating system, architecture, deadline, cost of execution and bandwidth. Apart from scheduling, a grid must ensure many aspects such as balanced load of resources, failure handling mechanisms, security of data and user satisfaction. These several independent issues make grid scheduling as a NP-complete problem [1].

The schedulers can be deployed level by level. The local scheduler is deployed within a cluster and is responsible for scheduling within the cluster. The scheduler at the top level is the grid broker. Scheduling can be centralized, decentralized and hierarchical. In centralized scheduling, the scheduler has more control over the resources. In decentralized scheduling, there is no central entity to have control over the resources and the scheduling decisions are made individually. In hierarchical scheduling, different levels of schedulers are deployed and scheduling is done at all the levels.

The proposed algorithm suits for computational grids with computing resources and scheduling is done by concentrating on load balancing, fault tolerance and several QoS requirements such as budget or cost and user deadline. The remaining part of this paper is organized with materials and methods which explain the works done previously with these factors and the newly proposed algorithm's architecture and nature. Then the experimental results are shown with comparisons and conclusions.

The grid computing environment comprises of heterogeneous resources which are distributed geographically. Hence, identification of a suitable resource for the submitted task is a tedious process. Many researchers have proposed algorithms for mapping of tasks to resources. Some of them

Dr.P.Keerthika is with the Computer Science and Engineering Department, Kongu Engineering College, Perundurai, Erode-638052, Tamilnadu, India (phone: 04294226560; e-mail: keerthikame@gmail.com).

Dr. P. Suresh is with the Information Technology Department, Kongu Engineering College, Perundurai, Erode-638052, Tamilnadu, India (phone: 04294226570; e-mail: sureshme@ gmail.com).

concentrate on user satisfaction, some on load balancing and some on fault tolerance.

An algorithm is proposed in [1] that begins with Min-min algorithm if the number of available resources is odd and starts with Max-min algorithm if the number of available resources is even. The remaining tasks are assigned to their appropriate resources by one of the two strategies, alternatively.

A Minimum Time to Release Scheduling Algorithm [2] has been discussed which depends on the time to release (TTR). It includes the processing time, waiting time and transfer time of input and output data to and from the resources. Based on the TTR value, the tasks are arranged in descending order and scheduled to resources with minimum TTR. This algorithm performs better when compared to First Come First Serve (FCFS) Scheduling and Min-min algorithms.

A Divided Min-min Scheduling algorithm [3] classifies jobs according to their ETC values as average, minimum and maximum. Then, it divides the jobs into same size segments and schedules the large job segment first and then the small job segment. It uses Min-min algorithm for scheduling. Different from Min-min, it sorts jobs before scheduling, which means that the job with long execution time will be scheduled earlier.

A fault tolerance service based on different types of failures satisfying the QoS requirements is proposed in [4]. It has a fault detector, fault manager, resource manager, resource allocation manager, meta computing directory service and execution time predictor. It allocates resources based on QoS requirements and performs job migration in case of occurrence of failures.

A Minimum Total Time to Release (MTTR) algorithm [5] reduces the time to release value by allocating computational resource based on job requirements, characteristics and hardware features of resources. It adopts a check pointing based fault tolerance and the check points are based on failure rate. It proposes a Replica Resource Selection Algorithm to provide checkpoint replication service.

In [6], the root cause of failures is studied from the real time data and categorizes them as human, environment, network, software and hardware. The failure rate are analyzed as a function of system and node and identified that the failure rates do not grow significantly faster than system size. Failure rate is analyzed at different time scales and statistical properties of time between failures are also defined.

The performance of most commonly used fault-tolerant techniques in grid computing is analyzed in [7]. The metrics such as throughput, turnaround time, waiting time and network delay are considered for evaluation. The average percentage of faults and the workloads are varied to analyze the behavior of these techniques. It analyses the task level fault tolerance mechanisms such as retrying, alternate resource, check pointing and replication.

The importance of fault tolerance for achieving reliability is surveyed [8] by all possible mechanisms such as replication, check pointing and job migration. It extends the cost optimization algorithm to optimize the time without incurring additional processing expenses. This is accomplished by applying the time-optimization algorithm to schedule task farming or parameter-sweep application jobs on distributed resources having the same processing cost.

In [9], a fault tolerant scheduling architecture that employs job replication is proposed. The algorithm determines adaptively the number of job replicas based on resource failure history. Then, it schedules the replicas to efficient resources using the backup resource selection algorithm.

A cost optimization scheduling algorithm is described in [10] to optimize the cost to execute the jobs. It optimizes time, keeping the cost of computation at minimum. 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 static heuristic approach [11] is proposed for scheduling independent tasks in grid environment which considers user satisfaction. The requirements of tasks are necessary to identify suitable resources. The proposed scheduling algorithm considers both system and application aspects i.e., the factors to improve the system performance and utilization of the resources and throughput. It makes use of the user deadline of tasks, data transfer time and the computation time for each <job, resource > pair for making scheduling decisions.

A grouping based scheduling algorithm [12] is proposed which considers user deadline and reduces communication time by adopting the grouping technique. The grouping strategy followed in this algorithm groups the fine grained tasks to coarse grained tasks based on the user deadline and computation time.

An efficient load balancing and grouping based job scheduling approach for grouping of fine-grained jobs is proposed in [13]. Its main goal is to maximize resource utilization and minimize processing time of tasks. It schedules tasks based on number of tasks available at a particular time and resource capability. Independent fine-grained jobs are grouped together based on the dynamically specified group and resource characteristics.

A neighbour level load balancing mechanism is proposed in [14]. A more accurate load measurement method is applied to determine the load of each resource. A load balancing algorithm is executed based on the information exchanged between neighbour nodes. If any node is overloaded then the load on every neighbour's node are evaluated and finds underloaded nodes. Then the task is shifted to underloaded nodes.

A hybrid load balancing policy which integrates static and dynamic load balancing technologies is proposed in [15]. Essentially, a static load balancing policy is applied to select effective and suitable node sets. It reduces 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 system level load balancing [16] is proposed where a distributed load balancing model transforms grid topology into a forest structure. A two level strategy is proposed to balance the load among resources of computational grid. In level 0, each cluster manager is associated with a physical cluster of the grid. The cluster manager is responsible for maintaining the workload information related to each one of its worker nodes, estimating the workload of associated cluster and diffusing this information to other cluster managers, deciding to start intra-cluster load balancing, sending the load balancing decisions to the worker nodes which they manage for execution and initiating the inter-cluster load balancing. In level 1, the worker nodes of a grid that are linked to their respective clusters are determined. Each node at this level is responsible for maintaining its workload information, sending this information to its cluster manager and performing the load balancing decided by its cluster manager. Load balancing schemes for grid environment [17] is proposed that does not follow the changes in the system status or set fixed threshold for controlling the load.

A dynamic and distributed protocol is designed in [18]. 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 coordinators.

A fault tolerant hybrid load balancing algorithm [19] is proposed which 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, then 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. The variation in site status can be identified at any of the cases when the grid scheduler receives the message that a certain site can no longer provide resources or when job execution on a certain site exceeds the expected execution time or when the site is overloaded.

A load balancing mechanism, which works in 2 phases, is proposed in [20]. 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 two 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.

A dynamic, distributed load balancing scheme for a grid environment is proposed [21] which provides deadline control for tasks. 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 based on deadline request and load. In [22], a hybrid algorithm is proposed for optimal load sharing with two components such as hash table and distributed hash table. It finds the nearest node and shares the load of a highly loaded node to lightly loaded node. It proves to provide the best tradeoff between space usage and lookup time. All these algorithms mentioned in literature concentrate on load balancing, fault tolerance and user satisfaction to an extent. But none of them considers all these factors combined. This research proposes a Budget Constrained Load Balancing Fault Tolerant Algorithm (BLBFT) which considers all these factors during scheduling. The architecture and the algorithm of BLBFT are explained below. In our previous work [23], 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. 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.

In [25], 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 Prioritized user demand algorithm is proposed [26] 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 [27] that consider 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.

II. Materials and Methods

A. Problem Formulation

The proposed algorithm follows a centralized scheduling architecture depicted in Fig. 1 where the scheduling is done only at the grid broker. Also it follows a static batch mode scheduling in which the tasks are scheduled in batches and when a task is allocated with a resource, it will not be changed. Hence the proposed algorithm is static, batch mode,

centralized scheduling algorithm.

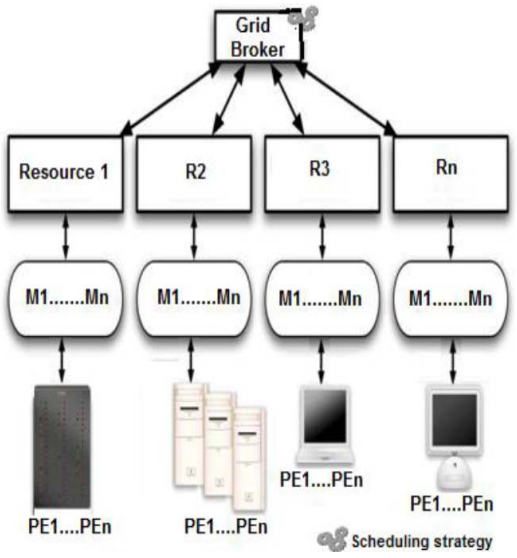

Fig. 1 Centralized Scheduling Architecture

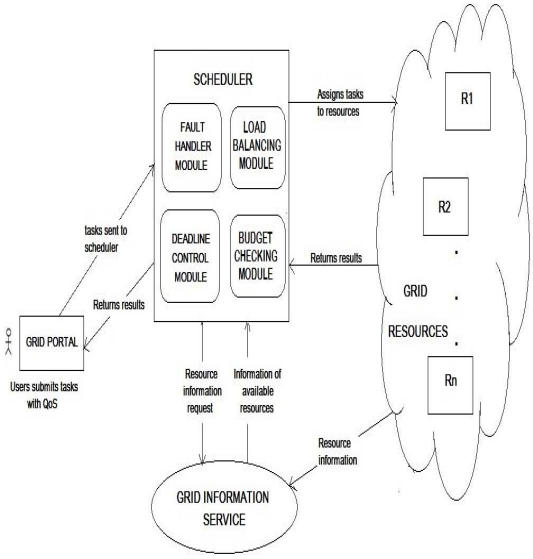

Fig. 2 BLBFT Architecture

B. Proposed BLBFT Scheduling Architecture

The scheduling architecture BLBFT algorithm is depicted in Fig. 2. The users submit the tasks to the grid broker through grid portal. The tasks are submitted along with the QoS requirements such as task completion deadline and execution cost. The grid portal submits the tasks to the grid scheduler/broker. The architecture has a Grid Information Service (GIS) which collects the information of all the resources involved in grid such as initial failure rate, number of tasks submitted, number of tasks successfully completed, availability time and processing capability in MIPS. The scheduler has four components.

The first is the fault handler module which calculates the failure rate of each resource and checks whether the selected resource has less failure rate. The second component is the deadline control module which takes care of user satisfaction in terms of satisfied deadline for task completion. The third is the load balancing module which updates the load of each resource and keeps control of balanced load. The fourth scheduler component is the budget control module which ensures minimized execution cost. This algorithm is implemented using the GridSim which follows the architecture depicted in Fig. 3.

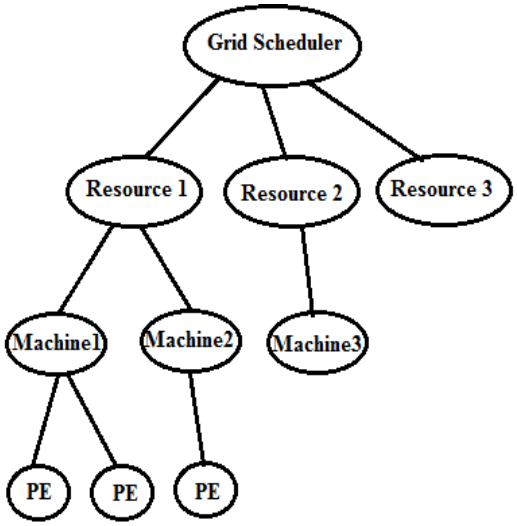

Fig. 3 GridSim Architecture

C. Proposed BLBFT Algorithm

The proposed BLBFT algorithm follows a static batch mode scheduling strategy in a centralized fashion. The algorithm is implemented at the grid broker level. It works as follows.

At the time of task submission to grid portal, the user submits the deadline and budget for task completion. The GIS receives the information of all the resources involved in grid such as computation cost. The algorithm makes use of these resource information and the user requirements and performs scheduling.

The load balancing module performs calculation of load and threshold value at all levels as follows. The load of each processing element is calculated by using the weighted sum of squares which is given by,

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

where L_k is the load attribute considered in our algorithm24. The load attribute considered in our algorithm is the CPU utilization in seconds. Hence the load of PE is given by,

$$Load(PE_i) = \frac{\sum_{j=0}^{n} MI_j}{MIPS_i} \quad (2)$$

where n is the number of tasks allocated to PE_i. The average load of each machine is calculated with the loads of PE's such as

$$AL(M_i) = \frac{\sum_{k=1}^{n} Load(PE_k)}{n} \quad (3)$$

where n is the number of PE's under Machine i . The average load of each resource is calculated by,

$$AL(R_i) = \frac{\sum_{k=1}^{n} AL(M_k)}{n} \quad (4)$$

where n is the number of machines under resource i.

The average load of the system/grid broker is calculated as,

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

where n is the number of resources in the system. After calculating the load of the resources, threshold value at the grid broker level is calculated as,

$$\Omega = AL + \sigma \quad (6)$$

where

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N} (AL\,(R_i) - AL)^2}{N}} \quad (7)$$

whereN is the number of resources in the system. In terms of gridsim the tasks are represented as gridlets.

The information of the gridlet such as gridlet size in million instructions (MI) is used to calculate the Expected Time to Compute matrix for all gridlets in all resources by using,

$$ETC(T_i, R_j) = \frac{Length_i}{Capacity_j} \quad (8)$$

The completion time matrix is calculated for each gridlet in each resource as,

$$CT(T_i, R_j) = ETC(T_i, R_j) + RT(R_j) \quad (9)$$

and the total completion time is calculated as,

$$TCT(T_i, R_j) = CT(T_i, R_j) + CMT(T_i, R_j) \quad (10)$$

The Budget control module calculates the cost matrix for executing each gridlet in each resource as,

$$CST(T_i, R_j) = ETC(T_i, R_j) \times CS(R_j) \quad (11)$$

The cost of execution and the expected budget from the user are compared and a suitable resource is selected.

The fault handler module calculates the failure rate of each resource with the information such as number of gridlets submitted and successfully completed. It is calculated using,

$$FR(R_j) = \frac{Tf}{Tsub} \quad (12)$$

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 resource j for execution. The ready time of each resource is calculated by,

$$RT(R_j) = \sum_{i=1}^{n} ETC(T_i, R_j) \quad (13)$$

where n is the number of tasks submitted toR_j.

TABLE I
SCHEDULING PARAMETERS AND THEIR VALUES

Parameters	Values
No. of Gridlets	512
Gridlet Length (MI)	50,000 to 1,00,000
I/P file size	50 to 500 MB
O/P file size	100 to 700 MB

III. RESULTS AND DISCUSSION

A. Experimental Setup

The proposed algorithm aims at reducing the makespan and to schedule efficiently with fault tolerance and balanced load. Also, the user satisfaction is considered with deadline control and budget parameters. The fault tolerance is ensured with improved hit rate and the user satisfaction is ensured with increased deadline hit count and reduced processing/execution cost. The balanced load is ensured with highest average resource utilization. Gridsim 5.0 toolkit is used for evaluating the proposed algorithm based on these factors.

- Number of Resources: 16
- Number of Tasks: 512

The gridlets assumed are independent, computationally intensive and arrive randomly and follows Poisson process. It is assumed that each resource can execute a single gridlet at a time. The scheduling parameters and the resource characteristics considered for scheduling are given in Tables I and II respectively.

B. Performance Metrics

The proposed algorithm is designed to satisfy the user with respect to deadline and budget, balanced load and fault tolerance. The performance metric used to evaluate the proposed BLBFT algorithm are makespan, hit count, deadline hit count, average resource utilization and execution cost. These performance metrics are defined below.

Makespan: This metric is for evaluating the overall performance of the scheduling algorithm. It is defined as the overall completion time of a batch of tasks and is given by,

$$Makespan = max\{RT(R_j)\}, \forall\ j \in n \quad (14)$$

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

TABLE II
GRID RESOURCE CHARACTERISTICS

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

Step 1: Get the list of tasks T from the user with their user deadline $UD(T_i)$ and Budget $B(T_i)$
Step 2: Get the list of resources R from GIS with the computation cost per second $CS(R_j)$ and initialize the deadline hit count and hit count values for all resources.
Step 3: Construct $ETC(T_i, R_j)$matrix of size m×n when m is the number of tasks and n is the number of resources.
Step 4: For all resources R_j in$R$, where 1≤ j ≤ n, and n denotes number of resources,
do
4.1: Calculate Failure rate
4.2: Calculate Ready Time
4.3: Calculate Load of each Processing Element using (1).
4.4: Calculate Average Load of each machine
4.5: Calculate Average Load of each resource
done
Step 5: Calculate Average Load of the system
Step 6: Calculate Balance Threshold
Step 7: Create a list of underloaded resources UR which has$AL(R_j) < \Omega$.
Step 8: For each task in T_i in queue and for each resourceR_j,
do
8.1: Construct $CT(T_i, R_j)$ matrix of size m×n
8.2: Construct $CMT(T_i, R_j)$ matrix of size m×n
8.3: Construct $TCT(T_i, R_j)$ matrix of size m×n
8.4: Construct cost matrix
done
Step 9: For all task $T_i$in Task_list$T$,
do
9.1: 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.
9.2: Select the resources in UT_{i_1} with $CST(T_i, R_j) \leq B(T_i)$ and create lists $UBT_{i_1}$and UBT_{i_2}. Include the list of resources in UT_{i_2}in UBT_{i_2}.
9.2: Sort the lists $UBT_{i_1}$and UBT_{i_2} based on $FR(R_j)$of resources in ascending order
9.3: Create lists $ULBT_{i_1}$and $ULBT_{i_2}$ with the set of underloaded resources from $UBT_{i_1}$and UBT_{i_2}respectively in order.
9.4: If entries in $ULBT_{i_1}$,
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 $ULBT_{i_2}$,
Select the first resource in the list for task T_i and dispatch $T_i$to resource $R_j$and Increment Hit Count.
9.5: Remove task $T_i$from Task_list T.
9.6: Update $RT(R_j)$and $FR(R_j)$ where j is the resource to which the task T_i is dispatched.
Done
Step 10: If there are tasks in Task_list$T$,
Repeat steps from 4.3.
else
Compute $Makespan = max\{RT(R_j)\}$ and
Compute $HitRate = \frac{T_{succ}}{T_{sub}} \forall j \in n$
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
$RU(R_j) = \frac{RT(Rj)}{Makespan} \times 100$
Compute Average Resource Utilization
$ARU = \frac{1}{N}\sum_{j=1}^{N} RU(R_j)$
endif

BLBFT Scheduling Algorithm

Hit count: Hit count is a new metric introduced that 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 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 can be calculated by (15):

$$RU(R_j) = \frac{\sum_{i=0}^{m} MI_i}{MIPS_j \times AT_j} \times 100 \quad (15)$$

The average resource utilization of the system can be calculated using (16):

$$ARU = \frac{1}{N}\sum_{j=1}^{N} RU(R_j) \quad (16)$$

where N is the number of resources.

Processing Cost: This metric is newly introduced in order to measure the algorithm's performance based on user satisfaction based on budget.

C. Experimental Results

The proposed BLBFT algorithm is compared with the Min-min algorithm which stands as a benchmark static heuristic algorithm for grid scheduling and the Fault Tolerant

Algorithm (FTMM) proposed in [25], Bicriteria Scheduling Algorithm (BSA) [23], LBFT algorithm [24] which is a load balancing algorithm for proving its performance based on makespan, hit count, deadline hit count, average resource utilization and cost.

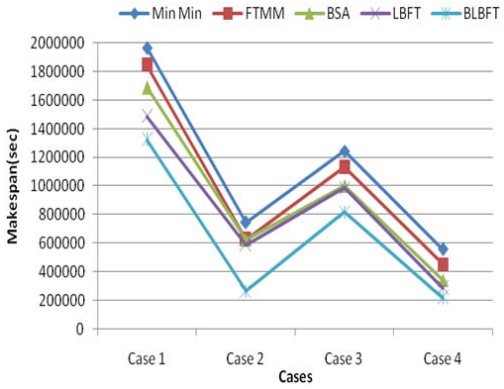

Fig. 4 Performance based on Makespan (Sec)

The performance comparison of the proposed BLBFT algorithm based on makespan is shown in Fig. 4. The results show that the BLBFT has minimized makespan than the other algorithms.

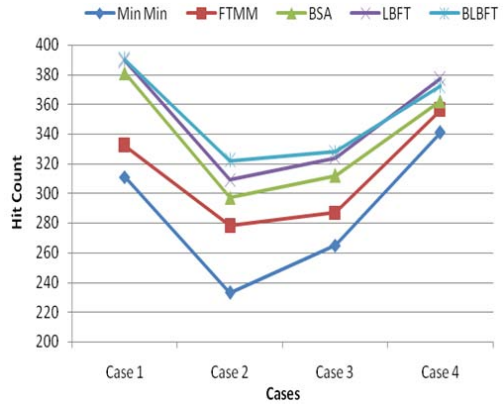

Fig. 5 Performance based on Hitcount

The performance of BLBFT based on hit count which is the measure of fault tolerance is shown in Fig. 5. The results show that the BLBFT algorithm has more number of gridlets successfully completed without failure.

The results of BLBFT based on deadline hit count are shown in Fig. 6. It is inferred that when compared with other algorithms such as Min-min, FTMM, BSA and LBFT, the proposed BLBFT has increased number of gridlets completed within user deadline.

The results based on resource utilization is shown in Fig. 7 and it is inferred that the proposed BLBFT algorithm has better resource utilization than the other algorithms such as Min-min, FTMM, BSA and LBFT which concentrates separately on each factor.

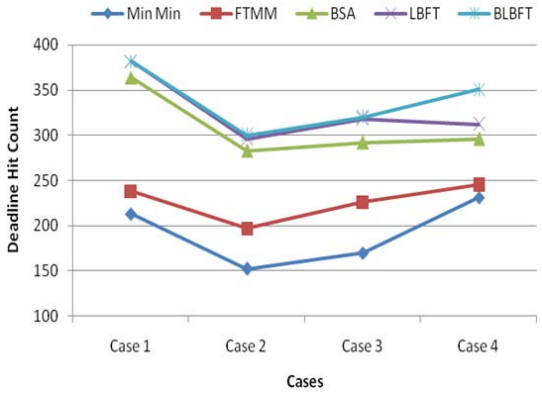

Fig. 6 Performance based on Deadline Hitcount

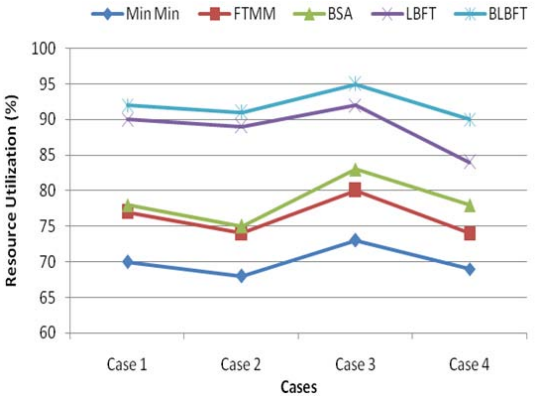

Fig. 7 Performance based on Resource Utilization (%)

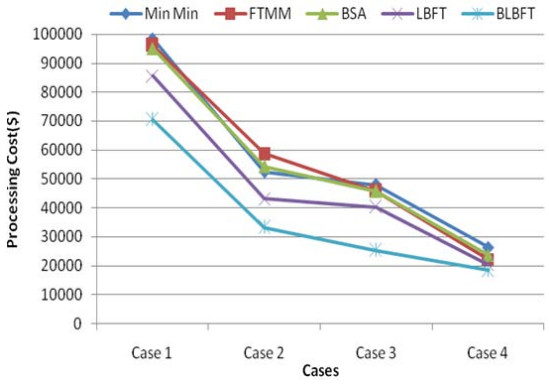

Fig. 8 Performance based on Processing Cost

The performance of BLBFT based on processing cost is shown in Fig. 8. The cost required to execute a batch of tasks is comparatively less for BLBFT than Min-min, FTMM, BSA and LBFT algorithms which do not concentrate on processing cost.

IV. Conclusions and Future Works

In this work, a budget constrained scheduling algorithm which mainly concentrates on processing cost is proposed. By reducing the processing cost, it makes an attempt to satisfy the user. Along with this cost factor, it also considers user

deadline of task completion to satisfy the user. With these two factors considered for user satisfaction, it also takes care of proper resource utilization and fault tolerance with reduced makespan.

The efficiency of this algorithm is proved by comparing it with already existing algorithms which separately concentrates on these factors based on makespan, hit count, deadline hitcount, resource utilization and processing cost. The applications considered in this work are computation intensive. In future, this can be extended for data intensive applications. This algorithm follows a centralized approach and in future, this can be extended in a hierarchical environment.

REFERENCES

[1] S. Parsa and R.E. Maleki, RASA: A New Grid Task Scheduling Algorithm, World Applied Sciences Journal, 7 (2009) 152-160.
[2] N. Malarvizhi and V.R. Uthariaraj, A minimum time to release job scheduling algorithm in computational grid environment, Proceedings of the IEEE Fifth International Joint Conference on INC, IMS & IDC, (2009) 13-18, DOI: 10.1109/NCM.2009.373.
[3] Z. Qian, Design of Grid Resource Management System Based on Information Service, J Computers, 5 (5) (2010) 687-694.
[4] 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, Proceedings of the IEEE International Conference on Computational Science and Engineering, (2009) 609-614, DOI: 10.1109/CSE.2009.257.
[5] M. Nandagopal and V.R.Uthariaraj, Fault tolerant Scheduling strategy for computational grid environment, Int J Engineering Science and Technology, 2 (9) (2010) 4361-4372.
[6] B. Schroeder and G.A. Gibson, A large-scale study of failures in high-performance computing systems, IEEE Trans Dependable and Secure Computing, 7(4) (2010) 337-350, DOI: 10.1109/TDSC.2009.4.
[7] F.G. Khan, K. Qureshi and B. Nazir,Performance Evaluation of Fault Tolerance techniques in Grid Computing System, J Computers and Electrical Engineering, 36 (6) (2010) 1110-1122, http://dx.doi.org/10.1016/j.compeleceng.2010. 04. 004.
[8] R. Garg and A.K. Singh, Fault Tolerance in grid computing: State of the art and open issues, Int J Computer Science & Engineering Survey, 2 (1) (2011) 88-97, DOI:10.5121/ijcses.2011.2107.
[9] M. Amoon, A development of fault- tolerant and scheduling system for grid computing, GESJ: Computer Science and Telecommunications, 3 (32) (2011) 44-52.
[10] 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, (2001) 24–27, http://arxiv.org/ftp/cs/papers/0203/0203020.pdf.
[11] P. Suresh and P. Balasubramanie, User demand aware scheduling algorithm for data intensive tasks in grid environment, European Journal of Scientific Research, 74 (4) (2012) 609-616.
[12] P. Suresh and P. Balasubramanie, Grouping based User Demand Aware job scheduling Approach for computational Grid, Int J Engineering Science and Technology, 4 (12) (2012) 4922-4928, http://www.ijest.info/docs/IJEST12-04-12-093.pdf.
[13] S. Kaur and S. Kaur, Efficient load balancing grouping based job scheduling algorithm in grid computing, Int J Emerging Trends and Technology in Computer Science,2(4)(2013) 138-144.
[14] M.A. Salehi, H. Deldari and B.M. Dorri, Balancing Load in a Computational Grid Applying Adaptive, Intelligent Colonies of Ants, Informatica, 33 (2) (2008) 159-167.
[15] K.Q. Yan, S.S. Wang, S.C. Wang and C.P. Chang, Towards a hybrid load balancing policy in grid computing system, Expert Systems with Applications, 36 (10) (2009) 12054-12064.
[16] B. Yagoubi and M. Meddeber, Distributed Load Balancing Model for Grid Computing, ARIMA Journal, 12 (2010) 43-60.
[17] K.S. Chatrapati,J.U. Rekha, and A.V. Babu, Competitive equilibrium approach for load balancing a computational grid with communication delays, J Theoretical and Applied Information Technology, 19 (2) (2010) 126–133.
[18] R.U. Payli, K. Erciyes, and O. Dagdeviren, Cluster-Based Load Balancing Algorithms for Grids, Int J Computer Networks & Communications, 3 (5) (2011) 253-269.
[19] J. Balasangameshwara and N. Raju, A hybrid policy for fault tolerant Load Balancing in grid computing environments, J Network and Computer Applications, 35 (1) (2012) 412-422, http://dx.doi.org / 10.1016 / j.jnca.2011.09.005,
[20] A.K. Bardsiri and M.K. Rafsanjani, A New Heuristic Approach Based on Load Balancing for Grid Scheduling Problem, J Convergence Information Technology, 7(1) (2012) 329-336.
[21] Y. Hao, G. Liu and N. Wen, An enhanced load balancing mechanism based on deadline control on GridSim, Future Generation Computer Systems, 28 (4) (2012) 657-665.
[22] D. Ramesh and A. Krishnan, Hybrid Algorithm for Optimal Load Sharing in Grid Computing, J Computer Science, 8 (1) (2012) 175-180.
[23] P. Keerthika and N. Kasthuri, An Efficient Grid Scheduling Algorithm with Fault Tolerance and User Satisfaction, Mathematical Problems in Engineering, 2013 (Article ID 340294) (2013).
[24] P. Keerthika and N. Kasthuri, A Hybrid Scheduling Algorithm with Load Balancing for Computational Grid, Int J Advanced Science and Technology, 58 (2013) 13-28.
[25] P. Keerthika and N. Kasthuri, An Efficient Fault Tolerant Scheduling Approach for Computational Grid, American J Applied Sciences, 9 (12) (2013) 2046-2051, Doi:10.3844/ajassp.2012.2046.2051.
[26] P. Suresh,P. Balasubramanie and P. Keerthika, Prioritized User Demand Approach for Scheduling Meta Tasks on Heterogeneous Grid Environment, Int J Computer Applications, 23 (1) (2011).
[27] P. Suresh and P. Balasubramanie, User Demand Aware Grid Scheduling Model with Hierarchical Load Balancing, Mathematical Problems in Engineering, 2013 (Article ID 439362) (2013).