Australian Journal of Basic and Applied Sciences, 9(20) June 2015, Pages: 632-639

ISSN:1991-8178

Australian Journal of Basic and Applied Sciences

Journal home page: www.ajbasweb.com

Cost Efficient Fault Tolerance Aware Resource Scheduling in Grid Environment

[1]K.Nirmala devi and [2]A.Tamilarasi

[1] *Research Scholar, Department of Computer Application, Kongu Engineering College, Perundurai, Erode, Tamilnadu, India*
[2] *Head of the Department, Department of Computer Application, Kongu Engineering College, Perundurai, Erode, Tamilnadu, India*

ARTICLE INFO

Article history:
Received 16 April 2015
Accepted 12 June 2015
Available online 26 June 2015

Keywords:
Grid computing, Task scheduling, Cost, Fault tolerance

ABSTRACT

Background: Grid computing is an upcoming developing technology which is used to distribute the resources across multiple sites that are present in different geographical locations for solving the computational problems. Scheduling and Resource management plays an important role in achieving high utilization of resources in grid computing platforms**. Objective:** The main objective of this work is to provision and schedule the jobs under checking the constraints called makespan, and cost. **Results:** In this method, to consider four types of mechanism for fault-tolerance strategies, including the job retry, the job migration without check pointing, the job migration with check pointing, and the job replication mechanisms. Simulation results show that the CFRTS algorithm has minimize the makespan and cost and it is more efficient for improving the job failure rate than the other existing algorithms. **Conclusion:** Findings demonstrates that the proposed methodology provides better results than the existing work in terms of improved system performance.

To Cite This Article: K. Nirmala devi and A. Tamilarasi., Cost Efficient Fault Tolerance Aware Resource Scheduling in Grid Environment. ***Aust. J. Basic & Appl. Sci.,*** *9(20): 632-639, 2015*

INTRODUCTION

Grid computing is the most popular trend in the distributed computing environment. The grid computing tends to share the resources across the sites which are distributed geographically with the consideration of reliability. There are various researches has been conducted to handle the different issues which may arise in the grid computing environment while sharing the resources across multiple sites. All universities, industries are intending to form the grid environment in their location privately because of its popular usage among multiple locations. In the recent years there has been number of researches are increased considerably which aims to provide the optimal resource provisioning. Scheduling is the one of the most importance task in the grid computing environment which need to be concentrated more for achieving the increased reputation level of cloud service providers. There are various researches has been conducted and accepted regarding the scheduling process. In the upcoming improved world, various methodologies need to be concerned more to improve the user satisfaction level and as well as the satisfying the QoS constraints that are mentioned by them. Efficient scheduling is done by matching the users QoS requirement with the available resources based on the user requirement.

This matching process of comparing the task requirement with the resource capacities are called as the mapping process which tends to allocate the task to the resource. Dynamic handlings of resources are most critical issue in the grid environment which needs to handle the tasks that are arriving into the grid environment in the dynamic manner. In the grid environment tasks are considered to be independent to each other for achieving the efficient scheduling through which QoS requirement of every task can be satisfied well. And also dynamic approach is introduced which aims to tolerate the incoming tasks that are arriving randomly in different time. This dynamic mapping needs to be done with the consideration of the available resource availability and the QoS requirements of users that are submitted by them. And also priorities need to be considered in order to achieve the better optimal allocation of resources for effective scheduling. This can be done introducing the methodology called the dynamic resource allocation scheme.

The mapping task in the grid environment can be done in three ways. Those are online mode, batch mode, and meta heuristic mode. In the online mode, tasks will be allocated immediately once it enters into the grid environment. Whereas batch mode

Corresponding Author: K.Nirmala devi, Research Scholar, Department of Computer Application, Kongu Engineering College, Perundurai, Erode, Tamilnadu, India

won't allocate the tasks as soon as it enters into the network. The tasks will be grouped together until obtains some utilization and then the group of tasks will be allocated. And in Meta heuristic mode, task will be allocated in the independent manner without affecting the processing of further tasks. However in Meta heuristic tasks wont start execution until all the tasks are grouped together. While online mode heuristics consider a task for mapping only once, batch mode heuristics consider a task for mapping at each mapping event until the task begins execution.

Local heuristics may not be sufficient to find an optimal solution to such types of problems. Meta-heuristic techniques are used to solve such types of problems efficiently. Meta-heuristic is an iterative master process that guides and modifies the operations of subordinate heuristics in order to produce high-quality solutions. This technique supports decision-making with robust tools that provide high-quality solutions to important applications in business, engineering, economics and science in reasonable time horizons. It also requires extensive knowledge in both problem domain and appropriate heuristic techniques. Moreover, meta heuristics are quite expensive to implement.

This work proposes a Cost-efficient fault tolerant resource aware task scheduling. This method represents different combinations of monetary cost and execution time for the same task. It is easy to see that none of the resource provisioning solutions is optimal with respect to both objectives. In this algorithm, when job scheduling consider four kinds of fault-tolerance mechanisms, including the job retry, the job migration without check pointing, the job migration with check pointing and the job replication mechanisms. This work showed that our algorithm provides a good balance between efficiency and monetary costs.

Related Works:

Online mode mapping heuristics:

In the on-line mode heuristics, each task is examined only once for matching and scheduling, i.e., the measuring is not changed once it is computed. When the arrival rate is low, machines may be ready to execute a task as soon as it arrives at the mapper. Therefore, it may be advantages to use the mapper in this mode so that a task need not wait until the next measuring event to begin its execution.

M.Poonguzhali *et al* (2012) introduced the MCT (minimum completion time) heuristic allows each task to the machine that results in that task's earliest finishing time. This purpose some tasks to be allocated to machines that do not have the minimum execution time for them. The MCT heuristic is used as a benchmark for the on-line mode, i.e., the performance of the other heuristics is compared against that of the MCT heuristic. As a task appears, all the machines in the GC suite are examined to determine the machine that gives the earliest completion time for the task. Therefore, it takes $O(m)$ time to map a given task.

Zikos, S *et al*., (2008) the MET (minimum execution time) heuristic allocates each task to the machine that achieves that task's computation in the least amount of execution time (this heuristic is also known as LBA (limited best assignment) and UDA (user directed assignment)). This heuristic, in difference to MCT, does not consider machine ready times. This heuristic can cause a severe imbalance in load across the machines. The benefits of this approach are that it gives each task to the machine that achieves it in the least amount of completion time, and the heuristic is very easy. The heuristic needs $O(m)$ time to find the machine that has the minimum execution time for a task.

In K. Hemant Kumar Reddy *et al*., (2012) the OLB (opportunistic load balancing) heuristic allocates the task to the machine that becomes ready next. It does not assume the execution time of the task when measured it onto a machine. If multiple machines develop into ready at the same time, then one machine is arbitrarily taken. The complexity of the OLB heuristic is dependent on the development. In the implementation assumed here, the mapper may need to examine all m machines to discover the machine that develops into ready next. Therefore, it takes $O(m)$ to find the assignment.

Batch mode mapping heuristics:

In batch mode, the mapper assumed as a meta-task for matching and scheduling at each mapping event. This enables the measuring heuristics to possibly make best decisions, because the heuristics have the resource requirement information for a whole meta-task, and know about the actual execution times of a larger number of tasks (as more tasks might complete while waiting for the mapping event). When the task arrival rate is high, there will be a sufficient number of tasks to keep the machines busy in between the examined events, and while a examined is being computed.

S. Lorpunmanee, Manoj Kumar, and S. Song *et al*., (2006) discusses the Min_min heuristic starts with the set U of all unassigned tasks. Then, the set of minimum finishing times,

$$M = \{\min_{0 \le j \le \mu} (ct(t_j, m_j)), \text{ for each } t_{i \in U}\}$$

Is found. Next, the task with the overall minimum finishing time from M is chooses and allocated to the corresponding machine (hence the name Min_min). Last, the newly mapped task is removed from U, and the action continuous until all tasks are mapped (i.e., U is empty). Min_min is based on the minimum finishing time, as is MCT. However, Min_min assumed as all unmapped tasks during each mapping decision and MCT only considers one task at a time.

S. Lorpunmanee, Manoj Kumar, and S. Song *et al*., (2006) discusses the Max_min heuristic is very similar to Min_min. The Max_min heuristic also

begins with the set U of all unallocated tasks. Then, the set of minimum execution times, M, is found. Next, the task with the overall maximum execution time from M is selected and assigned to the corresponding machine (hence the name Max_min). Last, the newly mapped task is removed from U, and the process repeats until all tasks are mapped (i.e., U is empty).

Chtepen, M and Krishnaveni (2014) the Duplex heuristic is accurately a mixture of the Min_min and Max_min heuristics. It makes both of the Min_min and Max_min heuristics and then uses the better solution. Duplex can be carried out to exploit the conditions in which either Min_min or Max_min performs well, with small overhead.

Krishnaveni *et al* (2012) the Sufferage heuristic is based on the scheme that enhanced mappings can be generated by allocating a machine to a task that would "suffer" mainly in terms of expected completion time if that particular machine is not allocated to it. Let the sufferage value of a task t_i be the variation between its second earliest execution time on some machine m_y and its earliest execution time on some machine m_x. Specifically, using m_x will outcome in the best completion time for t_i, and using m_y the second best.

Meta heuristic:

Krishnaveni (2012) A* is a search technique based on a μ-ary tree, starting at a root node that is a null result. As the tree grows, nodes correspond to partial mappings that is a subset of tasks is assigned to machines. The partial mapping (solution) of a child node has another task that is mapped than the parent node. Consider this additional task t_a. Every parent node generates μ children, one for each feasible mapping of t_a. Once a parent node has done this, the parent node turns into inactive. To keep completion time of the heuristic tractable, there is a pruning method to limit the maximum number of active nodes in the tree at any time.

Krishnaveni *et al*., (2013) the ant based algorithm is analyzed using the simulated execution times for a grid setting. Before starting the grid scheduling, the anticipated execution time for every task on each machine must be calculated and symbolized by an ET matrix. Each row of ET matrix contains the calculated execution time for a job on each resource and every column of the ET matrix is the calculated execution time for a specific resource of list of all jobs in the job team. ET_{ij} is the expected execution time of the machine m_j and task t_i. The time to shift the executables and data associates with the task t_i comprises the expected execution matrix ET_{ij}. For this algorithm, it is supposed that there are no inter-task contacts, each task can execute on each machine, and the estimated assumed execution times of each task on each machine is known as prior.

Proposed Work:

In workflow scheduling, different sub tasks of a bigger task are allocated resources in such a way that some pre-defined objective criteria are met. There are various problems in bioinformatics, astronomy and business enterprise in which a set of sub tasks is executed in a particular sequence in order to carry out a bigger task. In general, a workflow application requires series of steps to be executed in a particular fashion. These steps have parent child relationship. The parent task should be executed before its child task. The parent task is linked to child task according to set of rules. A workflow application is generally represented as a Directed Acyclic Graph (DAG) such as G (V, E) where V is the number of tasks and E is the information regarding data dependencies among tasks. A task which does not have any parent task is called entry task and a task which does not have any child task is called an exit task.

We define the workflow scheduling problem with respect to two simultaneous minimization objectives: makespan and monetary cost of the execution. We denote each individual schedule of a task i as a pair (i, j) meaning that the task i is assigned to resource j. We denote the schedule of the entire workflow as sched = {(i,j)|i ∈ [n]}.

The grid system consists of geographically dispersed computational sites having different administrative polices and heterogeneous resources. Any computational node may employ one or multiple fault-tolerance mechanisms for more reliable computation. Here, we consider the following four fault-tolerance mechanisms.

Job retry (JRT) mechanism:

The JRT mechanism is the simplest fault-tolerance technique, which will re-execute the failed job from the beginning on the same computational node.

Job migration / Job migration without check pointing (JMG) mechanism:

The JMG mechanism will move the failed job to another computational node and re-execute the job from the beginning on the latter computational node.

Job migration with check pointing (JCP) mechanism:

The JCP mechanism will record the state of the job periodically at rum time. If the job fails, it is moved to another computational node and resumed the execution from the last checkpoint.

Job Replication (JRP) mechanism:

The JRP mechanism replicates a job to multiple computational nodes such that the job has higher success rate. If one of those replicas has already completed, then all other replicas should stop their execution to save the computing power.

In the grid system, each computational site supports one of the following three mechanisms:

JRT, JMG and JCP. As for the supporting of JRP, the scheduler will allocate multiple computational sites to execute a certain job concurrently. Furthermore, the scheduler can execute a certain job by any combination of these four different fault-tolerance mechanisms. For instance, a job may be executed concurrently in a node supporting JRT as well as a node supporting JCP, resulting in that JRP is also applied to the job in effect.

The objective value is calculated using the following expression.

Minimize

$$\alpha \times T(i,\ j) + \beta \times C(i,\ j) for all r_j \in R$$

Subject to

$$T(i,j) = \frac{1}{\max_{j=1}^{p}\{\sum_{\forall i \in S_j} E(T_j^i)\}} \quad (1)$$

$$C(i,j) = \frac{c(C_j^i) - c_{min}}{c_{max} - c_{min}}, \alpha + \beta = 1 \quad (2)$$

where α is a cost-efficient factor that represents the user's preference for the execution time and the monetary cost; T (i,j) and C (i,j) represent cost-efficiency ratios of time and costs, respectively; $c_{min(max)}$ are, respectively, the minimum (maximum) monetary cost of any task scheduling plan. Then S_j represents the set of jobs assigned to Node j, and $E(T_j^i)$ represents the expected execution time for Job i running in Node j. Furthermore, $E(T_j^i)$ is calculated depending on which faulttolerance mechanism is adopted by Node j, which will be described in the following subsection.

In particular, we use a linear pricing model. In the linear pricing model, the cost of using a resource is linearly correlated with the number of CPU cycles. Let ca_{slow} denote the CPU cycle for the slowest R_{rslow}. If Vc_{base} is the base price charged to rslow, then the cost incurred to execute job T_i on R_j can be calculated as follows:

$$C_j^i = \sigma \times T(i,j) \times Vc_{base} \times (\frac{ca_{Rj}}{ca_{slow}}) \quad (3)$$

The total monetary cost is computed by

$$C = \sum_{\forall i \in S_j} C_j^i \quad (4)$$

where σ is a random variable used to make various combinations of resource pricing and capacity. In the selection step, select the resource based on the objective of (1).

Execution time Calculation:

The T(i,j)is calculated by Eq. (2), as mentioned in the above subsection, where the expected execution time is required. In this subsection, we explain how to calculate the expected execution time depending on which mechanism a job is protected.

JRT mechanism:

For the JRT mechanism, we calculate the expected execution time as follows. Calculate the $E(T_j^i)$ represents the expected execution time for Job i running in Node j

$$E_{RT}(T_j^i) = (1 + \frac{1}{2}p_j^i + \frac{1}{2}p_j^{i2} + \frac{1}{2}p_j^{i2}) \times \frac{SZ_i}{C_j} \quad (5)$$

Where, SZ_i is the size of Job i, and C_j is the computing capacity of Node j.

JMG mechanism:

The JMG mechanism is similar to the JRT mechanism. However, when a job fails, the job is migrated to another computational node for more secure computation, which imposes extra overhead. The calculation of the expected execution time for the JMG is as follows. Suppose the corresponding population is (i, j, k, q). The expected job execution times of the job i in nodes j, k and q are computed as follows.

If node a to j fails job is migrated to another computational node $E(T_j^i)$ represents the expected execution time for Job i running in Node j, k, q

$$E_{MG}(T_j^i) = (1 - \frac{1}{2}p_j^i) \times \frac{Sz_i}{c_j} + p_j^i \times MC_{j,k}^i \quad (6)$$

$$E_{MG}(T_k^i) = p_j^i \times ((1 - \frac{1}{2}p_k^i) \times \frac{SZ_i}{c_k} + p_k^i \times MC_{k,q}^i) \quad (7)$$

$$E_{MG}(T_q^i) = p_j^i \times P_k^i ((1 - \frac{1}{2}p_q^i) \times \frac{SZ_i}{c_q}) \quad (8)$$

Where

$$MC_{x,y}^i = \frac{D_i}{Bw_{x,y}} \quad (9)$$

$MC_{x,y}^i$ is the migration cost of the condition that Job i moves from Node x to Node y, D_i is the data size of Job_i, $Bw_{x,y}$ and is the communication bandwidth between Node j and Node j and Node k, where x; y∈{j,k,q}.

JCP mechanism:

The JCP mechanism is similar to the JMG mechanism. However, the JCP mechanism must store the transient process states to the check pointing server periodically. When a failure occurs, the check pointing server will restore the process state to the backup node before resuming the unfinished job. The computation of the expected execution time for JCP is as follows. Suppose that the corresponding gene is (i, j, k, q) and then the expected job execution times of the job i in nodes j, k and q are computed as follows.

If node i to j fails job transient process states to the check pointing server periodically the process to backup node before resuming the unfinished job

$$E_{cp}(T_j^i) = (1 - p_j^i)(\frac{SZ_i}{C_j} + (\frac{\frac{SZ_i}{C_j}}{PR}) \times OH_j) \quad (10)$$

$$RM_j^i(j,k,q) = SZ_i - \left(\left(\frac{\frac{SZ_i}{2\times C_j}}{PR}\right)\times PR\times C_j\right) \quad (11)$$

$$E_{cp}(T_k^i) = P_j^i\left(\left(1-P_j^i\right)\left(\frac{RM_j^i(j,k,q)}{C_k}\right)+\left(\frac{\frac{RM_j^i(j,k,q)}{c_k}}{PR}\right)\times OH_k\right) + p_k^i\left(\frac{RM_j^i(j,k,q)}{2\times C_k}\right) + \left(\left(\frac{\frac{RM_j^i(j,k,q)}{2\times C_k}}{PR}\right)\times OH_k + MC_{kq}\right) \quad (12)$$

$$RM_k^i(j,k,q) = RM_j^i(j,k,q) - \left(\left(\frac{\frac{RM_j^i(j,k,q)}{2\times C_k}}{PR}\right)\right)\times PR\times C_k \quad (13)$$

$$E_{cp}(T_q^i) = P_j^i\times P_k^i\left(\left(1-P_j^i\right)\left(\frac{RM_k^i(j,k,q)}{C_q}\right)+\left(\frac{\frac{RM_k^i(j,k,q)}{C_q}}{PR}\right)\times OH_q\right) + P_q^i\left(\frac{RM_k^i(j,k,q)}{2\times C_q}\right)+\left(\left(\frac{\frac{RM_k(j,k,q)}{2\times C_q}}{PR}\right)\times OH_k\right) \quad (14)$$

where, $RM_k^i(j,k,q)$ is the remaining job size for Job i to be executed when a failure occurs in Node x. OH_x is the overhead of performing one check pointing operation for Node x.

JRP mechanism:

The expected execution time for the JRP mechanism is calculated Let the set RP_i consists of those nodes that will execute Job i independently.

Let the set RP_i consists of those nodes that will execute Job i independently. Assume Job i starts to be executed in Node j at time s_i^j if Node j belongs to the set RP_i. If Job i is executed successfully, then the job will be finished at time $f_j^i = s_j^i + \frac{SZ_i}{c_j}$. Because the execution of job i in Node j will continue after time f_j^i only if all previous executed replicas fail, the probability that Job i will continue after time f_j^i will be:

$$P_{const_i(f_i^j)} = \prod_{w\in RP_i} p_w^i \quad (15)$$

Let the execution time of each replica is broken into multiple pieces by f_j^i where jϵ RP_i. Each piece has an execution probability and its expected execution time is equal to multiplying the continuation probability at the beginning of a piece by the execution time of executing Job i in Node j is calculated as follows

$$E_{RP}(T_j^i) = P_{start}(s_j^i)\times(-s_j^i) + \sum_{\substack{x,j\in RP_i \\ s_j^i\le f_{xy}^i\le f_j^i}} P_{consti}(f_x^i)\times(f_y^i - f_x^i) \quad (16)$$

Where

$$s_j^i < f_u^i, \neg\exists f_v^i, s_j^i < f_v^i < f_u^i \quad (17)$$

and

$$P_{starti(s_j)} = \begin{cases} 1 \text{if} \neg\exists f_a^i, f_a^i < s_j^i \\ P_{cont_i}(f_h^i) \text{if} s_j^i \ge f_h^i, \neg\exists g, s_j^i > f_g^i > f_h^i \end{cases} \quad (18)$$

Algorithm 1: Cost-efficient fault tolerant resource aware task scheduling:

Input: G (N, E): the DAG task graph

1. Initialize the number of resources R= {r_1,…,r_n}n number of resources and T={t1,…,tn}j is the n number of jobs.
2. For each ti ϵ T do
3. For each ri ϵ R do
4. Compute α* T (i, j) +β * C (i, j)
5. End
6. End
7. For each task ti in the ready queue do
8. Assign task ti to the resource rj that minimizes the objective function equation 1 of task ti
9. end

Assign the tasks to the resources based on Pareto dominance. Successive tasks are mapped to the resources based on the constraints. For each task, choose the resource that favors scheduling tasks with low monetary cost to run it. This is done by the pre-defined objective function (1).

Experimental Result:

The experimental tests were conducted to compare the effectiveness of the proposed methodology by comparing it with the existing approaches. In this section, the performance is validated to compare the existing MGSO with MBAT algorithm and the proposed Cost Efficient Fault tolerant based scheduling algorithm. The following are the graphical results of our implemented systems namely CFTRS considered for the comparison of these methods are namely

- Response time
- Co-ordination delay and
- Makespan
- Cost

Based on the comparison and the results from the experiment shows the proposed work works better than the other existing systems with higher fault tolerance rate

A. Response Time:

Response time for a task is the delay between the submission time and the arrival time of execution output which is shown in Fig.1 Effectively, the

Australian Journal of Basic and Applied Sciences, 9(20) June 2015, Pages: 632-639

response time includes the latencies for coordination and the CPU time.
Response time is calculated as follows:
Response Time = Entering time + Schedule start time

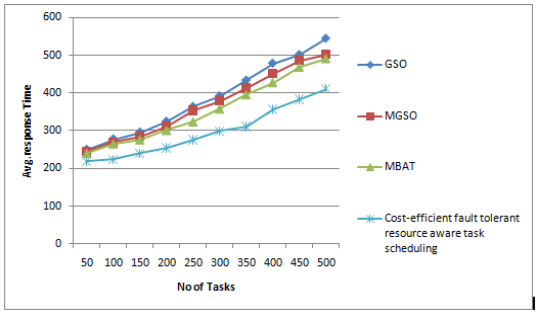

Fig. 1: Average Response Time Graph.

In Fig., Number of tasks ranging from 50 to 500 is taken along x-axis and average response time per task (in seconds) is taken along y-axis ranging from 0 to 600. It can be inferred from the graph that response time of Cost-efficient fault tolerant resource aware task scheduling is lesser than BAT+ Resource Provisioning, MBAT, MGSO and GSO, which shows Cost-efficient fault tolerant resource aware task scheduling is more responsive than BAT+ Resource Provisioning MBAT, MGSO and GSO.

B. Average Co-Ordination Delay:

The metric coordination delay sums up the latencies for: (i) resource claim to reach the index cell, (ii) waiting time till a resource ticket matches with the claim, and (iii) notification delay from coordination service which is shown in Fig.2.

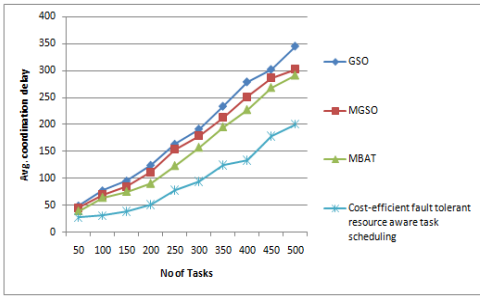

Fig. 2: Average Coordination Delay Graph.

The metric coordination delay sums up the latencies for: (i) resource claim to reach the index cell, (ii) waiting time till a resource ticket matches with the claim, and (iii) notification delay from coordination service to the relevant GAM which is shown in Fig. on which number of tasks ranging from 50 to 500 is taken along x-axis and average coordination delay time per task (in seconds) is taken along y-axis ranging from 0 to 350. It can be inferred from the graph that coordination delay time of Cost-efficient fault tolerant resource aware task scheduling is lesser than BAT+ Resource Provisioning, MBAT, MGSO and GSO which shows Cost-efficient fault tolerant resource aware task scheduling is more coordinating than BAT+ Resource Provisioning, MBAT, MGSO and GSO.

C. MakeSpan:

Makespan is measured as the response time of a whole workflow, which equals the difference between the submission time of the entry task in the workflow and the output arrival time of the exit task in that workflow which is shown in Fig.3.

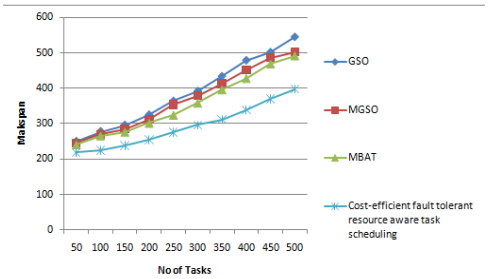

Fig. 3: Makespan Graph.

Makespan is measured as the response time of a whole workflow, which equals the difference between the submission time of the entry task in the workflow and the output arrival time of the exit task in that workflow which is shown in Fig.. Note that, these measurements (except makespan) are collected by averaging the values obtained for each task in the system. The measurement of makespan is taken by averaging over all the workflows in the system. In Fig., Number of tasks ranging from 50 to 500 is taken along x-axis and average makespan workflow is taken along y-axis ranging from 0 to 500. It can be inferred from the graph that makespan of Cost-efficient fault tolerant resource aware task scheduling is lesser than BAT+ Resource Provisioning, MBAT, MGSO and GSO which shows Cost-efficient fault tolerant resource aware task scheduling is quicker in completion of workflow than BAT+ Resource Provisioning, MBAT, MGSO and GSO.

D. Cost:

Cost is used to compare the profit which is obtained by executing the tasks which are submitted by the users. The cost comparison is given as follows:

The costs associated with a particular resource can vary depending on the time period when the resource is used. By performing a detailed analysis of the resource costs during one or more specific periods, you can determine when the optimum times are for scheduling activities that require the resource. You can also use this analysis in combination with resources costs summary analysis to compare the costs of different resources, which enables you to set resource requirements using the resource that has the

lowest cost. In Fig., Number of tasks ranging from 50 to 500 is taken along x-axis and cost workflow is taken along y-axis ranging from 0 to 500. It can be inferred from the graph that cost of Cost-efficient fault tolerant resource aware task scheduling is lesser than BAT+ Resource Provisioning, MBAT, MGSO and GSO which shows Cost-efficient fault tolerant resource aware task scheduling is quicker in completion of workflow than BAT+ Resource Provisioning, MBAT, MGSO and GSO.

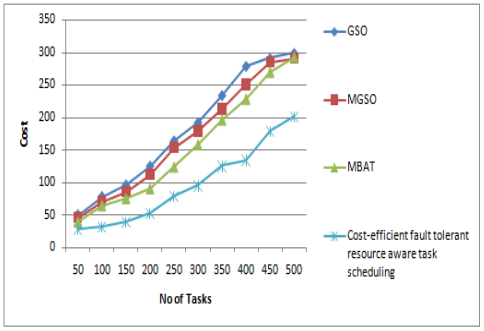

Fig. 4: Cost Graph.

Conclusion:

In this work propose a new task-scheduling algorithm for running large programs in the grid. Most of task scheduling algorithms do not consider monetary costs, and so they cannot be directly applied in a grid setting. In this proposed algorithm computes scheduling plans that produce makespan as good as the best known algorithm of while significantly reducing monetary costs also deal with fault tolerant mechanism. In this method, to consider four types of mechanism for fault-tolerance strategies, including the job retry, the job migration without check pointing, the job migration with check pointing, and the job replication mechanisms. We evaluated our algorithm using gridsim. Simulation results show that the CFRTS algorithm has minimize the makespan and cost and it is more efficient for improving the job failure rate than the other existing algorithms.

REFERENCE

Liang Hu, Xi-Long Che and Si-Qing Zheng, 2012. ” Online System for Grid Resource Monitoring and Machine Learning-Based Prediction” in IEEE transactions on parallel and distributed systems, 23(1).

Poonguzhali, M., 2012. ” Dwelling-Time Based Resource Scheduling Algorithm Using Fuzzy Logic in Grid Computing ” in International Conference on Computer Communication and Informatics (ICCCI - 2012), 10-12.

Zikos, S., H.D. Karatza, 2008. Resource allocation strategies in a 2-level hierarchical grid system. In: Proceedings of the 41 st Annual Simulation Symposium (ANSS), 13-16, IEEE Computer Society Press, SCS, 157-164.

Hemant Kumar Reddy, K. and Diptendu Shina Roy, 2012. “A Hierarchical Load Balancing Algorithm for Efficient Job Scheduling in a Computational Grid Testbed” in Recent Advances in Information Technology.

Lorpunmanee, S., M.N. Sap, A.H. Abdullah, C. Chompoo-inwai, 2007. “An Ant Colony Optimization for Dynamic Job Scheduling in Grid Environment.” International Journal of Computer and Information Science and Engineering, 207–214.

Manoj Kumar, Mishra Prithviraj Mohanty, G.B. Mund, 2012. “A Time-minimization Dynamic Job Grouping-based Scheduling in Grid Computing.

Song, S., Y.K. Kwok, K. Hwang, 2006. Risk-resilient heuristics and genetic algorithms for security-assured grid job scheduling, IEEE Transactions on Computers 55(6): 703-719.

Chtepen, M., F.H.A. Claeys, B. Dhoedt, F. De Turck, P. Demeester, P.A. Vanrolleghem, 2009. Adaptive Task Checkpointing and Replication: Toward Efficient Fault-Tolerant Grids Parallel and Distributed Systems, IEEE Transactions on, 20(2):180-190.

Krishnaveni, S., M. Hemalatha, 2014. “Evaluation of DFTDS algorithm for distributed data warehouse”, Egyptian Informatics Journal, 15: 51-58.

Krishnaveni, S., Hemalatha, 2012. “M. Query processing in distributed data warehouse using proposed dynamic task dependency scheduling algorithm”, Int J Comput Appl, 55(8): 17-22.

Krishnaveni, S., M. Hemalatha, 2012. “Query scheduling in distributed data warehouse using DTDS and VMFTRS algorithms”, Eur J Sci Res., 89(4): 612-25.

Krishnaveni, S., M. Hemalatha, 2013. “Query management in data warehouse using virtual machine fault tolerant resource scheduling algorithm”, Int J Theor Appl Inform Technol, 47(3): 1331–7.

Larson, J.W., M. Hegland, B. Harding, S. Roberts, L. Stals, A.P. Rendell, P. Strazdins, M.M. Ali, C. Kowitz, R. Nobes, J. Southern, N. Wilson, M. Li, Y.Oishi, 2013. “Fault-Tolerant Grid-Based Solvers: Combining Concepts from Sparse Grids and MapReduce”, ICCS Procedia Computer Science, 130-139.

Bungartz, H.J., M. Griebel, 2004. Sparse grids, Acta Numerica, 13: 147–269.

Chauhan, P., Nitin, 2012. “Fault Tolerant Decentralized Scheduling Algorithm for P2P Grid”, 2nd International Conference on Communication, Computing & Security [ICCCS], 698-707.

Chauhan, P., Nitin, 2012. “Decentralized Computation and Communication Intensive Task Scheduling Algorithm for P2P Grid”, 14th International Conference on Computer Modelling and Simulation (UKSim), 516.

Keerthika, P. and N. Kasthuri, 2013. "An Efficient Grid Scheduling Algorithm with Fault Tolerance and User Satisfaction", Hindawi Publishing Corporation, Mathematical Problems in Engineering.

Chauhan, P., Nitin, 2013. "Fault Tolerant PLBGSA: Precedence Level Based Genetic Scheduling Algorithm for P2P Grid", Hindawi Publishing Corporation, Journal of Engineering.

Asgarali Bouyer, Mohd Noor MD SAP, 2013. "A Prediction-based Fault Tolerance on Grid Resources scheduling by using Optimized Case-based Reasoning", comp.utm.my.

Zahra Pooranian, Mohammad Shojafar, Jemal H. Abawajy and Mukesh Singhal, 2013. "GLOA: A New Job Scheduling Algorithm for Grid Computing", International Journal of Artificial Intelligence and Interactive Multimedia, 2-1.

Kiruthika, P. and S. Gokul Dev, 2014. "Dynamic Scheduling and Rescheduling With Fault Tolerance Strategy in Grid Computing", IJAICT, 1-2.

Behnam Barzegar, Habib Esmaeelzadeh and Hossein Shirgahi, 2011. "A new method on resource management in grid computing systems based on QoS and semantics", Indian Journal of Science and Technology, 4(11): 1416-1419.