Analysis of Fault Tolerance on Grid Computing in Real Time Approach

Er.Parampal Kaur, Er. Deepak Aggarwal

Abstract-- In computational Grid, fault tolerance is an imperative issue to be considered during job scheduling. Due to the widespread use of resources, systems are highly prone to errors and failures. Hence fault tolerance plays a key role in grid to avoid the problem of unreliability. Scheduling the task to the appropriate resource is a vital requirement in computational Grid. The fittest resource scheduling algorithm searches for the appropriate resource based on the job requirements, in contrary to the general scheduling algorithms where jobs are scheduled to the resources with best performance factor. The proposed method is to improve the fault tolerance of the fittest resource scheduling algorithm, by scheduling the job in coordination with job replication when the resource has low reliability. Based on the reliability index of the resource, the resource is identified as critical. The tasks are scheduled based on the criticality of the resources. Results show that the execution time of the tasks is comparatively reduced with the proposed algorithm using real time approach rather than simulator.

Keywords: Computational Grid, Fault tolerance, Job Scheduling, Task Replication.

1.INTRODUCTION

Fault tolerance is an important property for large scale computational grid systems, where reliability and availability, the grid infrastructure should be a foolproof fault tolerant. Since the failure of resources affects job execution fatally, fault tolerance service is essential to satisfy QOS requirement in grid computing. Commonly utilized techniques for providing fault tolerance are job check pointing and replication. Both techniques mitigate the amount of work lost due to changing system availability but can introduce significant runtime overhead. The latter largely depends on the length of checkpointing interval and the chosen number of replicas, respectively. In case of complex scientific workflows where tasks can execute in well defined order reliability is another biggest challenge because of the unreliable nature of the grid resources. In the grid environment the resources are heterogeneous and highly distributed. Hence they are prone to failures. Any scheduling algorithm will be more effective if fault tolerance is taken into account. The idea in this paper is to enhance the effectiveness of MFRS algorithm proposed by Ruay et.al [15] by adding the fault tolerance feature to the algorithm. In the grid computing environment, when a resource is over exploited that is, if a resource is allocated to many Tasks, it depletes the power of the resources leading towards its failure. The execution time also increases for the job which is assigned to these resources.

To solve this problem, the less reliable resources are identified as critical resource. The identification of the resources as critical resource is done on the basis of failure rate of the resource. If the resource failure rate is increasing gradually and retains constant then the resource is critical. geographically distributed nodes co-operate to execute a task. In order to achieve high level of

For all the resources which are less reliable, when a task is allocated to it, there are greater probability of failure which can cause severe problems for the users of the Grid environment. For this, Task which are assigned to those critical resources are replicated to another resource in the same level. This duplicated task is now again kept on the Task List for execution. So the chances of the successful execution of the submitted Task will increase. The resource which fails for a particular time is not assigned to any other job during that period. The failed jobs are again added to the Task List and are ready for the scheduling again. These jobs are allocated to the same level resource if any resource is present in same level or they are allocated with the resource of next level. This concept of replicating of the job will increase the reliability. And also reduce the overhead of the resources. The fault tolerance is the imperative issue of this paper. The failures are predicted as transient failures which are randomly generated and for a random interval of time. The failed resource will be again available for the allocation once the problem of the resource is rectified.

2. RELATED WORK

Grid resource scheduling is one of the most significant research issues today. Most of the existing scheduling algorithms do not consider the resource failure during scheduling which would eventually increase the execution time of the task. Some of the resources scheduling algorithms which are more related to the issues considered in the paper are discussed along with their shortcomings.

Metacomputing prototype Charlotte uses the eager scheduling mechanism [3] where the tasks are assigned to the free nodes randomly to keep them busy. There is a manager process which schedules the jobs and volunteers, who volunteer to execute the job. The parallel jobs having concurrent routines are split and given to different volunteers for execution. When all the routines are allocated and still if volunteers are free the unfinished jobs will be reassigned to the volunteers. If a machine fails, the job is migrated and reexecuted in another machine automatically. The advantage of this mechanism is that even if there is one machine which runs normally for a long period, there is higher probability that all jobs will be executed successfully. The failure of a machine or a slow running machine will not affect the execution of the jobs. But the problem is that same task may be replicated in many resources such that newly arrived tasks do not have any available nodes which reduce the system throughput. Replicating the task in many resources possibly increases the probability of successful execution of the task, but in turn replicating the tasks, when there is insufficient resources, delays the job execution.

D.Saha et.al.,[4] describes the FPLT algorithm in which the scheduler sorts the node and task information by each node's CPU speed and task's workload in order to reduce complexity for searching the fastest node and the largest task all the time. Then the scheduler assigns the largest task of the waiting tasks to the available fastest node. FPLTF will reduce to the same as Workqueue when the tasks arrive one by one Wang et.al [18] in their MFTF algorithm declares the fitness between the task and the node based of the expected execution time and execution time of the node. The node which has much less difference is expected to be the fittest node and the task is allocated to it. The fitness definition seems to be arbitrary.

Bajaj and Aggarwal [17] proposed an algorithm TANH (Task duplication-based scheduling Algorithm for Network of Heterogeneous systems) in which a new parameter is introduced for each task: the favorite processor (fp), which can complete the task earliest. Other parameters of a task are computed based on the value of fp. In theclustering step, the initial task of a cluster is assigned to its first fp, and if the first fp has already been assigned, then to the second and so on. Duplication based algorithms are very useful in Grid environments. The computational Grid usually has abundant computational resources (recall that the number of resource is unbounded in some duplication algorithms), but high communication cost. This will make task duplication very cost effective. Most of the dynamic scheduling algorithms are based on predictions based on historical records. Yang et al [11] provide three levels of prediction. The first level is the simplest one-step-ahead prediction in CPU load changing and historical peak values are used as thresholds. If the performance of current time point is increasing, the performance for the next time point will be predicted as increasing unless the upper threshold is reached, and vice versa. Based on this one point prediction, interval performance prediction for the near future can be obtained by aggregating multiple point performance in previous intervals and using the one-step-ahead prediction scheme to the aggregated value. The variance of performance in an interval can also be obtained in a similar way: using historical data of a series points to get the standard deviation of the performance during each interval, and then applying the one-step-ahead scheduling scheme to standard deviations of previous intervals. A machine with lower interval variance is considered more “reliable”, and a scheduler “conservatively” assigns less work to high-variance resources [5].

Conservative scheduling [11] assigns less work to less reliable resource. This algorithm avoids the wave of delayed behavior caused by variance in the resource capability. Here the conservative load prediction is done based on the predictive mean and predictive variance. Conservative scheduling technique uses the predicted mean and variance of CPU capacity to make data mapping decisions. The basic idea is to allocate more work to systems that which are expected to deliver the most computation, where this is defined from the viewpoint of the application. A resource with a larger capacity will also show a higher variance in performance and therefore will more strongly influence the execution time of an application than will a machine with less variance. A cluster may be homogeneous in machine type but quite heterogeneous in performance because of different underlying loads on the various resources.

3. Fault Tolerant MFRS Algorithm

The most fitting resource algorithm [15] proposed the closeness factor which describes the appropriateness of the resource with the job requirements. The scheduling algorithm schedules the job to the appropriate resource rather the best performing resource. In the MFRS[15] algorithm the resources are categorized into L discrete levels. The literature entire resource is divided into ten levels and the fittest resources are allocated to the nodes from these levels. These levels are generated dynamically by calculating the PET. In this paper a new parameter called Reliablity index is proposed to improve the fault tolerance of the most fitting resource scheduling algorithm. As we know when workload increases then the chances for the job failure increases. So once the resource fails the jobs allocated to that resource fails. If a resource fails continuously for a number of times then we can’t be sure

that the Task submitted to that particular resource will execute successfully. This problem will lead to the wastage of time as well as the resource and will increase the average waiting time for the Task. To get rid of this problem the reliability of the node is measured at the time of scheduling. If the most fitting node has less reliability level, the task is replicated to node in the same level. The reliability of the node is measured as follows.
Let f be the number of failures in given time say t.

$$f = \frac{\text{no.of failures}}{\text{total time of execution}} \qquad (1)$$

When there is no failure $(1 - f)^t$
$(1 - f)^t = e^{-ft}$
Taking log
$t \log (1 - f) = - fr \log e$
Differentiating with respect to t
$\frac{1}{f} = \log\frac{e}{\log(-f)=a}$
So $f = \frac{1}{a}$
Node’s reliability at time t
$NR_t = e^{-ft} = e^{\frac{t}{2}}$
Where $a = \frac{\text{Total Execution Time}}{\text{No of failures}}$

The tasks that are allocated to the less reliable resources are replicated and a new ID is given to them with the same parameters is added to the Grid list. The Grid user will be more secure if the Task given by him is successfully executed. Once the Tasks are failed it is added back to the Grid list. This will reduce the risk of job failure. In this real time approach only the transient failures of the resources has been considered. Failure of resource is generated randomly at random interval of time. These resources become available after a certain interval of time and can be used as normal resources. This concept of cloning of the job will increase the reliability of the grid environment. Providing a fault tolerant grid environment improves the efficiency of the system.

Algorithm

Step 1: The task Ti is submitted for scheduling.
Step 2: The resources are categorized into different discrete levels based on the predicted execution time for the given task Ti.
Step 3: The resources are marked Critical based on the Reliability Index value.
Step 4: The closest level Li Based on the desired execution time of the Task Ti is identified.
Step 5: Any node Ri within the Level Li with minimum load is allocated to the Task Ti.
Step 6: If the resource Ri is marked Critical, then Task Ti is replicated and submitted to
another resource Rj with minimum load in the same level Li.

4. Results Comparison

4.1 Present Work

This section shows comparison between real time approach using socket programming and Java simulator. In this section our main focus on the task execution speed. The speed to execute any task is increased with real time approach as compared to java simulator. We have shown comparison between real time approach and java simulator with the help of following tables. Table shows execution speed and CPU utilization, on the basis of these term we have identified the performance. Totally 10 resources ranging from 1 to 10 were created as per the specifications in table. Each of these resources are divided into 10 different levels after the calculation of Predicted Execution Time (PET)[15] for each job. The average waiting time of each resource is also calculated. Each resource has its own queue and all the Tasks coming is added to these queues

TABLE 1
Performance of real time approach

CPU ID	CPU Speed	CPU Utilization
1	2 GHz	0.5
2	2.2 GHz	1.1
3	2.5 GHz	1.3
4	2.7 GHz	1.0
5	2.9 GHz	1.0
6	3.1 GHz	0.3
7	3.3 GHz	1.8
8	3.5 GHz	1.9
9	3.7 GHz	0.7
10	3.9 GHz	0.5

after scheduling the job. Once the resource is assigned to any Task it can’t be allocated to other Task until it freed. Each resource has its own queue which holds the Task assigned to it. Following table shows performance of real time approach:

We have concluded from the table that CPU speed or execution speed of task consistently increases as compared to simulator. The previous simulator performance will be discussed in the next section.

4.2 Previous Work

The Java based Simulation environment similar to the one created by Ruay et.al [15] for implementing MFRS algorithm was created and used to analyze the performance of the FMFRS algorithm.

In the following table, we have present the performance of java based simulator created by A. Shamila Ebenezer and K. Baskaran[1].

TABLE 2
PERFORMANCE OF JAVA BASED SIMULATOR

CPU ID	CPU Speed	CPU Utilizaion
1	550 MHz	0.3
2	1GHz	0.1
3	1.1GHz	0.1
4	1.2 GHz	0.2
5	1.3GHz	0.4
6	1.4 GHz	0.3
7	1.5GHz	0.8
8	2 GHz	0.5
9	2.3 GHz	0.6
10	2.5 GHz	0.5

The next part of the module is the creation of the Tasks. These Tasks are created by the Grid users. Two Grid users are created by using socket programming. Each user is capable of creating specific numbers of tasks. Each Task has certain parameters including workload, requested speed and closeness factor. The Tasks also include a unique ID and a Boolean parameter Success to state that where the job is executed successfully or not. Workload is given in millions of instructions and its range varies from 300000 to 500000. Normally if the workload is very high then it is tough to fulfill all the requirements. The workload for each Task is generated randomly. The requested Speed is given in millions of instruction per second (MIPS). This depicts the power of resource needed to accomplish the task. The requested speed is randomly generated in the range of 1000 to 4000 (MIPS). If have another parameter closeness factor this parameter is generated in the range of 0.1 and 1.0 and is used for deciding the levels from which the resource has to be allocated .The Desired Execution Time (DET) [15] is calculated for each gridlet coming to the simulator by using this equation 3.2.

DET=Wj/Rj (2)

Where,
DET= Desired Execution Time
Wj= workload
Rj=requested Speed

The user is modeled to submit 100 tasks for every 100 seconds. Failures are considered to be transient.

5. PERFORMANCE ANALYSIS

The Tasks are scheduled using FMFRS algorithm for several iterations and for each iteration, the execution time was observed to be less using the FMFRS algorithm. One such iteration is displayed using graph with task in x axis and CPU speed in Y axis.

The fault tolerant MFRS scheduling is analyzed in real time approach and found that the execution speed was less with simulator as compared to the real time approach. Comparatively the performance of the MFRS algorithm with fault tolerance is observed to be effective.

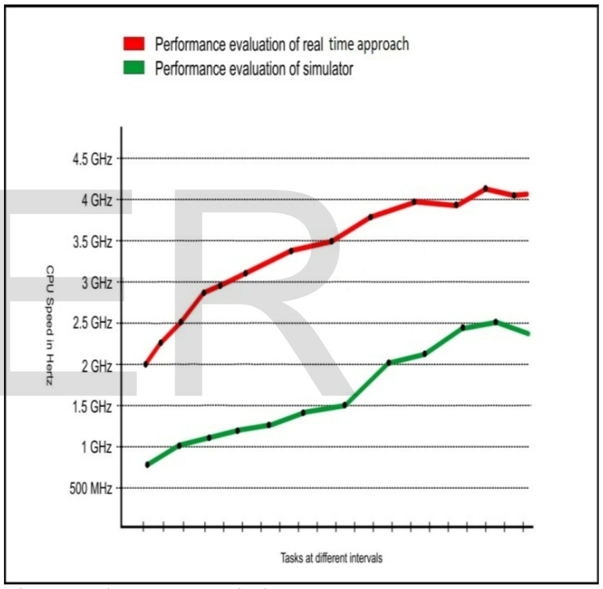

Fig. 1. Performance Analysis

CONCLUSION & FUTURE SCOPE

Resource allocation has become a battle with the acceleration in number of resources especially in dynamic environment. The participating resources in a grid may be computational resources, data storage or computer network. The fault tolerant most fitting resource scheduling was implemented in a java based grid environment and the results are evaluated and concluded that with fault tolerance feature added to the scheduling algorithm, the turnaround time of the task is comparatively very less.

The algorithm was tested for transient failures and our future scope would be to test the algorithm for other failures like crash and omission failure.

ACKNOWLEDGMENT

I would like to thank Dr. Lakhwinder Singh, Coordinator, PTU Regional Center, Baba Banda Singh Bahadur Engineering College, Fatehgarh Sahib for kind support and encouragement he provided to me during this research work. I also owe my sincerest gratitude towards Prof. Deepak Aggarwal for his valuable advice and healthy criticism throughout my work which helped me immensely to complete my work successfully. It was only because of his constant guidance and support that, I was able to complete my paper. I feel indebted to him for the same. I would like to thank the members of the Departmental Research Committee for their valuable suggestions. Their inputs have been instrumental in improving the quality of my work. I would also like to thank everyone who has knowingly & unknowingly helped me throughout my paper work. Last but not least, a word of thanks for the authors of all those books and papers which I have consulted during prepration of my paper work. At the end thanks to the Almighty for strength and patience he gave me to take my work to a logical end.

REFERENCES

[1] A. Shamila Ebenezer "Fault Tolerant most Fitting Resource Scheduling Algorithm (FMFRS) for Computational Grid" *European Journal of Scientific Research ISSN* 1450-216X Volume: 86 No 4 September, 2012, pp.468-473 © EuroJournals Publishing, Inc. 2012.

[2] Abdul Aziz, Hesham El-Rewini," Grid Resource Allocation and Task Scheduling for Resource Intensive Applications".

[3] Baratloo, M. Karaul, Z. Kedem, P. Wyckoff,"Charlotte: Metacomputing on the Web,"*Proceedings of the 9th International Conference on Parallel and Distributed Computing Systems*, 2003.

[4] D. Saha, D. Menasce, S. Porto, et al., "Static and dynamic processor scheduling disciplines in heterogeneous parallelarchitectures", *Journal of Parallel and Distributed Computing* 28 (1)(1995) 1–18.

[5] F. Dong and S. G. Akl, "Scheduling algorithm for grid computing:state of the art and open problems", *Technical Report of the OpenIssues in Grid Scheduling Workshop*, School of Computing, UniversityKingston, Ontario, January, 2006

[6] Foster. I and Kesselman, C.: "The Grid2: Blueprint for a new computing infrastructure",*Elsevier Inc.*, Second Edition, 2004.

[7] Georgiana marin "Grid Computing Technology" *Database Systems Journal* vol. II, No. 3/2011.

[8] Herbert Schildt, "Java 2: The Complete Reference", Fifth Edition, *2002-08-13*

[9] J.Jaybharathy and Ayeshaa Parveen.A,"A Fault Tolerant Load Balancing Model for Grid Environment", *Internatinal Journal of Recent trends in Engineering*, Vol 2, No.2,162-164,2009.

[10] Liu Chong, and Lu Huapu, "Study of Traffic Information Analysis and Decision Support System Based on Grid Computing", *International Journal of Information Technology*, Vol. 12 No.6 2006.

[11] L. Yang, J. M. Schopf and I. Foster, "Conservative Scheduling: Using Predicted Variance to Improve Scheduling Decisions in Dynamic Environments", *in Proc. of the ACM/IEEE SuperComputing2003 Conference*, pp.31-- 46, Phoenix, Arizona, USA, November 2003.

[12] Malarvizhi Nandagopal, V.Rhymend Uthariaraj, "Fault Tolerant Scheduling Strategy for Computational Grid Environment", *International Journal of Engineering Science and Technology*, Vol.2(9), 4361-4372, 2010.

[13] O. Ardaiz, P. Artigas, T. Eymann, F. Freitag, L. Navarro, and M. Reinicke, "The catallaxy approach for decentralized economic-based allocation in grid resource and service markets," *Applied Intelligence*, vol. 25, no. 2, pp. 131–145, 2006.

[14] Ritu Garg " Fault Tolerance in Grid Computing:State of Art and Open" *International Journal of Computer Science & Engineering Survey* (IJCSES) Vol.2, No.1, Feb 2011.

[15] Ruay-Shiung Chang, Chun-Fu Lin, Jen-Jom Chen,"Selecting the most fitting resource for task execution" *Future Generation Computer Systems*, Volume 27, Issue 2, february 2011 ,pages 227-231.

[16] S.Baghavathi Priya and Dr.T.Ravichandran" Fault Tolerance and Recovery for Grid Application Reliability using Check Pointing Mechanism" *International Journal of Computer Applications* 26(5):32-37, July 2011.

[17] S. Ranaweera and D. P. Agrawal, "A Task Duplication Based Scheduling Algorithm for Heterogeneous Systems", *in Proc. of 14th International Parallel and Distributed Processing Symposium (IPDPS'00)*, pp. 445-450, Cancun, Mexico, May 2000.

[18] S. Wang, I. Hsu, Z. Huang , "Dynamic scheduling method for computational grid Environments" *, in: Proceedings of the International Conference on Parallel and Distributed Systems*, July 2005, pp. 22–28.