A PREDICTIVE APPROACH TO IMPROVE A FAULT TOLERANCE CONFIDENCE LEVEL ON GRID RESOURCES SCHEDULING

Asgarali Bouyer[1], Mohd Noor Md Sap[1]

[1]Faculty of Computer Science and Information Systems
University Teknologi Malaysia
81300 Skudai, Johor

Email: [1]{basgarali2@siswa.utm.my, Mohdnoor@utm.my}

Abstract: The grid is becoming more attractive and encouraging platform for solving large-scale computing intensive problems. In this environment, various geographically distributed resources are logically coupled together and presented as a single integrated resource. Since resources on grid are dynamic and [farar], one of the most important problems in grid environment is design a fault tolerance resources scheduling. Therefore, finding a stable and fault tolerance resource require designing a predictive method that doing this work. Many methods are presented in a few years ago, but in these algorithms, some parameters such as job requirements and clear predictor method are not truly considered and also some methods apply optimistic view in grid scheduling cycle. On the other hand, since many methods use from GIS information to learn about resources, so they cannot powerfully select optimal nodes because the GIS don't cover all information about grid resources. Due to this disadvantage, this paper presents a new approach on fault tolerance mechanisms for the resource scheduling on grid by using Case-Based Reasoning technique in a local fashion. This approach applies a specific structure in order to prepare fault tolerance between executer nodes to retain system in a safe state with minimum data transferring. Certainly, this algorithm increases fault tolerant confidence therefore, performance of grid will be high.

Keywords: Grid, Resource scheduling, Fault Tolerance, Case-Based Reasoning, Scheduler

1. INTRODUCTION

Grid computing [5, 6] is an amazing infrastructure to solve some problems that need to strong and heavy computation with very long time execution. It is cooperation of different computers, for a specific task, so that the user acquires better performance for that specific

task. In this environment, the resources are geographically distributed, but in logical aspect, these are as virtual single resource with high performance. In other word, grid computing is a distributed computing system that several group of computers are connected to create and work as one large virtual computing power. Grid computing allows a group of computers to share the system securely and optimizes their collective resources to meet required workloads by using open standards OGSA (Open Grid Services Architecture) [7]. The Grid allows executing jobs in different nodes. In order to perform job scheduling and resource management at Grid level, usually it has used a Resource scheduler or a meta-scheduler. A scheduler is fundamental in any large-scale Grid environment. The task of a Grid resource scheduler is to dynamically identify and characterize the available resources, and to select the most appropriate resources in order to submit jobs. In grid scheduling discussion, selecting best nodes with looking at economic and fault tolerance criteria is considerable [6]. Choosing the suitable fault tolerance resource for a user job to meet predefined constraints such as deadline, speedup and cost of execution is an important problem in the grids. In our approach, we highly have solved some of these problems.

As known, grid scheduling consists of three steps. The first step is resource discovery and filtering, and the second is selecting nodes and scheduling jobs to related nodes, and the last step is submitting and monitoring jobs. Surely, step 2 is vital because some nodes always have best behavior, while some others often have fault with low performance [12, 13].

Resources scheduling is one of the most important sections in grid because if resources have not been scheduled, we have many failures, very high time execution and low accuracy. The performance of the grid depends strongly on the efficiency of the scheduling. A schedule is an assignment of jobs to a set of resources. Each job can be divided to set of tasks, and one of the grid resources can execute each task for a certain time.

In scheduling phase on grid, schedulers usually use some information about resources' attributes (CPU speed and load, memory) to do the scheduling. The information used by the schedulers is usually provided by an information service that is responsible for gathering data about all resources that compose the grid. Key problem is that information obtained from the GIS may be out of date by the time the scheduler needs it to schedule tasks. In our approach, we have used an online scheduling with novel information.

Since, grid is a dynamic and heterogeneous environment, so a failure can occur every time and in every node, independent of other nodes in grid. Essential to fault tolerance is the recovery from an error. An error is that part of the system that may lead to a failure. The whole idea of error recovery is to replace erroneous state with an error-free state. We used an optimal fault tolerance approach by applying an optimized case-based Reasoning (OCBR) algorithm [1] to prediction, detection and recovery of faults in grid because OCBR is one of

the preferred problem-solving strategies and machine learning techniques in complex and dynamically changing situations.

The rest of this paper is organized as follows. Section 2 gives an overview on previous research in fault tolerance resource scheduling. Section 3 describes an optimized CBR that is used in our method. Section 4 discusses the system design and implementation details of our Grid resource scheduling respectively. Section 5 describes experimental results and section 6 concludes the paper.

2. RELATED WORKS

In recent years, many researchers have offered many methods, frameworks or algorithms for dynamic job scheduling in different notions. The most objective was failure nodes problem in task scheduling. Unfortunately in the grid, the possibility on failure in resource node is not deniable. In the past, many fault tolerance job scheduling has been suggested for grid or cluster computing [14, 13, 3, 2].

S. Baghavathi Priya and et al. presented a fault tolerance approach for Task Scheduling by using Genetic algorithm on grid [13]. They have used checkpoint technique in their method that is a general-purpose method for providing fault tolerance in distributed systems. Check pointing allows the recovery to a previous correct state. In their approach, a new Genetic algorithm is offered to get a better fault tolerance. This Genetic Algorithm can accomplish both evolving the system from a random arrangement to the near-perfect solution and final perfect optimal solution. Due to simplicity and understandability, their method is a good one. Nevertheless, for a large-scale computing with thousands nodes, surely this algorithm take much time. In addition, they have considered some assumptions for scheduler, which require more investigation.

GRIDTS [3] is another infrastructure for Fault-Tolerant Scheduling in Grid environment. This proposed approach allows scheduling decisions to be made with up-to-date information about the resources. GRIDTS provides fault-tolerant scheduling by combining a set of fault tolerance techniques to deal with crash faults in components of the system. Fault tolerance in GRIDTS is enforced using a combination of mechanisms. Check-pointing is used to limit the work lost when a resource fails during the execution of a task, allowing another resource to continue the task from the last checkpoint, where the task was left unfinished. Their approach does not use GIS information, because GIS information might be abrogated.

B. Nazir and T. Khan present another approach that is simply performable [2]. They developed a new method for fault tolerant job scheduling in grid. Proposed approach maintains history of the fault occurrence of resource in Grid Information Service (GIS). Whenever a resource broker has job to schedule, it uses the fault occurrence from history

information (GIS), and depending on this information, it uses different intensity of check pointing and replication while scheduling the job on resources that have different tendency towards fault. In addition, their approach has used check-pointing technique to create a reliable and efficient system. They claimed that it increases the percentage of jobs executed within specified deadline and allotted budget, hence helping in making grid trustworthy. However it is possible that this algorithm cannot be optimal, because data in GIS might be old and abrogated.

There are some other related works such as [9, 8, 10, and 11] that we devolve them to readers.

3. OCBR ALGORITHM

This method applies CBR algorithm by using Decision Tree in order to select suitable sampling. OCBR consists of two phases:

- Phase 1: primary processing of information to make Decision Tree and assigning each existing record (or sample) to its related class. Decision Tree in this research is used to select a best training set in appropriate branch, that coming job exists in those branches, for CBR algorithm [4] and also for mining of useful Rules.
- Phase 2: final processing and predicting the situation of a record (coming job), by using neighboring records.

At the first step, to identify the main and efficient parameters in existing database system and to clarify their effect on final result, we do a processing operation on the obtained results. Then we try to classify the information into different classes (by using decision tree classifier). At the second step, first we try to insert the desired record into its class according to previously done classification in Decision Tree. Then considering the number of desired neighbors, we select the existing records that are similar to our desired record and perform the predicting operation (CBR algorithm).

Then at the beginning, the information or primary system parameters are identified and integrated. Next, we can get the final result of the problem by performing the final processing among the desired record and its neighbors (in the same class).

4. SYSTEM ARCHITECTURE

In this approach, there is a local database for every node in grid that is considered to store some useful information about submitted jobs and status of execution. When a new job is submitted on a node, then a new recode will be inserted to its database. In addition, we have considered a queue on grid scheduler that is called Reservation Queue, to support faulted nodes. When one of the nodes in site is failed, if reservation queue is not empty, then a new

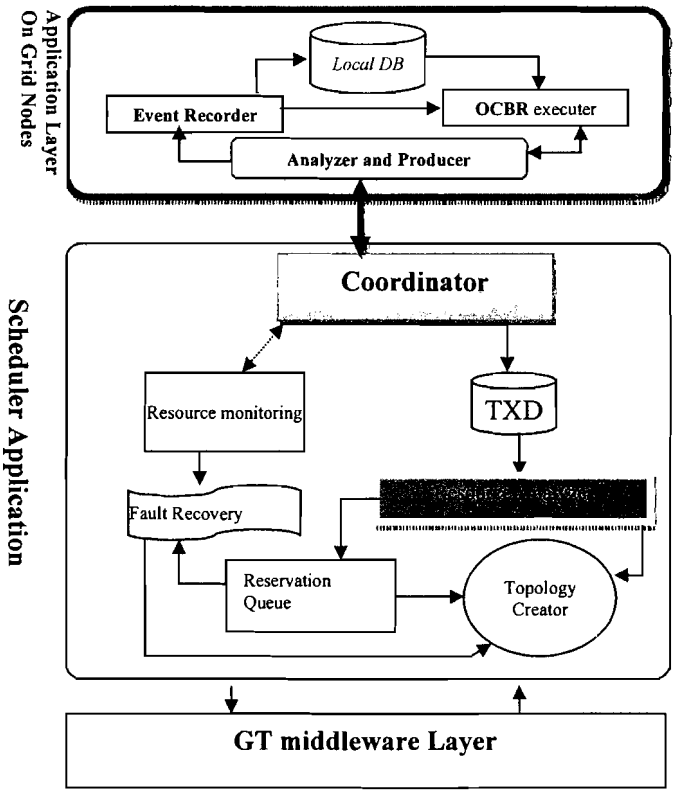


Figure 1: An architecture for fault tolerance grid scheduling.

node replace with faulted node. Scheduler is responsible to this change node, because that is aware from status of nodes in site by using message passing. As we mentioned, grid scheduling had three step that step 2 was very important. This approach will act after step 1.

We have shown a general architecture for this approach (figure 1). For the nonce, we have provided an isolated application that must be installed and executed on each node as well as another application that will be installed on scheduling machine for this purpose.

4.1. Scheduler's Application

This application is responsible to select fault tolerance nodes and doing a fault tolerant cycle to retain system in a tolerant safe state. For implementing a fault tolerance scheduling, we shall do following phases:

Phase 1: coordinator is responsible to send a packet to every node on grid. This packet include some information about new coming job and a request for executing a Optimized Case-Based Reasoning (OCBR) [1] in local database by desired node based-on this coming job. Scheduler wants to find nodes with the least fault and best performance in past. At the end,

the produced results will be sent to coordinator to be saved in a temporary XML database (TXD).

Phase2: coordinator will insert all received results from each node in a XML database. After that, Resource Analyzer will analyze this results based-on fault tolerant criteria and job condition by executing several queries to get better and squarer decision. For example, it finds a node that has best fault tolerance for small jobs but it is not suitable for large-scale jobs. Therefore, the scheduler must consider coming job status in its decision or selection.

At the next, Resource analyzer selects the best fault tolerance nodes to start operations and extra nodes will reserve in a queue that we called it Reservation Queue. Also for each node it will consider a priority.

Phase 3: creating a Virtual topology in a Ring form, based on the nearest neighbor on the right side of each node -as near as possible and without any node repetition. Always scheduler will be created according to node's priority one example of which is given in figure 2. Now, if there is a fault happens in one of the nodes, Fault Recovery is called to resolve it. At the beginning, Fault Recovery will check Reservation Queue to find a node for replacing with failed node, but "what will it do if the queue is empty?"

Since the job belonging to failed nodes may be an important job with high priority, so this job should not be left unfinished and scheduler is forced to run it. Therefore, Fault Recovery section will send a message to left-hand of faulted node in order to create a connection with right-hand of faulted node and its job transfer to right-hand node to continue this unfinished job there. Of course, we do not want to apply many nodes for reservation, because, we have developed this approach based on economic grid. As you know, in the real world, nobody likes others to use his\her computer free. Therefore, it is very important to design a fault tolerant computation on grid with minimum computational resources.

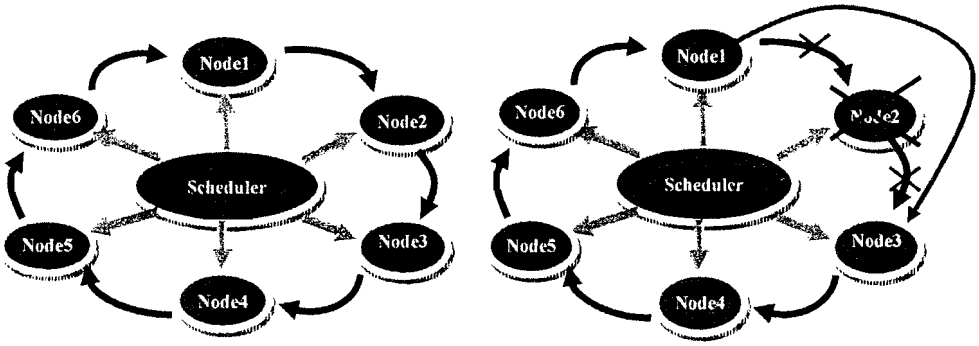
4.1.1 Proposed Topology

When scheduler had selected desired nodes, it will create a topology in the form of a ring. An important point that must be mentioned is that in this topology, each node is communicating with right side node and these two nodes are the nearest possible neighbors. In other words, all nodes have a near neighbor on their right-hand. If a node failed, with the assumption that queue is empty, scheduler will do as follows:

Step1: Fault Recovery Section scans cause of failure. If the desired node has a hardware problem it will do step 2. If the node is alive, or connected to grid, but it cannot continue the related job for the reason such as CPU-Idle RAM or Secondary Storages problem, then Fault Recovery Section will define a random time, called chance time to revival, in order to revive

probability. If in this allocated time, the desired node has started again, it needn't do any operation. Otherwise, it must start step 2.

Step2: At this step, Fault Recovery Section will transfer the job of failed node to right-hand node. For example, figure 2 represents a site with six nodes. As you see, in figure 2-(a) system is in safe state. Let's assume that node 2 suddenly finds a hardware problem, and also Reservation Queue is empty, then this section simply removes it and transfer its job to the right-hand node (figure_2-(b)). If transferred job has a real-time priority, then desired node can stop its job and start this real-time job, but its job also had a Real-time priority then it can transfer previous job to next node. For getting a better performance we can use checkpoint



(a) (b)

Figure 2. The management of failed nodes by ring fashion. As you see, (a) show nodes communication before fault, and (b) show the failed node2.

technique [13] in while of executing job.

Since the discovered node in grid is very much, therefore, we will not have lack of computing resources. Nevertheless, it is likely that the problem pay fee for powerful and reliable resources such as Super-computers or Cluster-computers have been existed. But, surely we don't have problem in using a usual resource computing, for example Personal computers (PCs).

4.1.2 Scheduling Based On Multi Versioning

Fault tolerance is an important aspect of grid resource management. Our another provided objective is to propose grid-scheduling algorithm with high reliability and high fault tolerance through using multi versioning. The term Version of a job in this paper refers to a copy or replica of the job that will be sent to another machine. The objective is to execute job versions in parallel form to raise fault tolerance. Because grid resources are dynamic and

heterogeneous, the error and fault occurrence are unavoidable. A Failure can occur every time and in every node, independent of other nodes in grid. Grid scheduler has not deterministic information about arrival time of jobs in the future, service time of jobs, memory requirements and processor requirements, only scheduler might have an estimation of that information, and therefore, grid scheduler makes online decisions. As mentioned before, there are many other nodes that have low priority and thus, they will stay in queue. We can use these nodes in order to support primary selected nodes. In this approach, every Primary Selected Node for executing desired job can have Protector Nodes that execute a version or replica of job belongs to supported node. Both Primary Node and Protector Node will execute desired job in parallel mode. Needless to say, the low priority nodes will always have Protector Nodes. If one of replicas is terminated, to avoid resource wastage, coordinator section in scheduler will close all nodes that are executing the same replicas of job in grid and release its related nodes. Then, these nodes will be inserted to Reservation Queue.

On the other hand, if one of the Primary nodes, without having a Protector Node, fails and Reservation Queue is empty, then the faulted node will replace with one of the other protector nodes (related to other jobs).

Grid scheduler periodically checks the status of jobs and their versions. If there is a fault declaration in a running version from a node in a site, grid scheduler tries to keep job deadline, then grid scheduler immediately changes the Node and shifts its focus on other versions.

4.2. Node's Application

For better prediction, we have provided a specific Node's Application (NA) for every node in a purposed grid. NA contains an internal local database that considered for recording all events about submitted jobs by grid and only Event Recorder section can write in database. When a new job accepted and submitted on node, or while a job are completed or failed, this section will record all of mentioned events. This section is more important.

There is a section that we called 'Analyzer and Producer'. As mentioned above, before determining fault tolerant nodes, scheduler sends a packet that includes information about coming job (e.g. IP Sender, Size of the job, Size of needed RAM and HDD, average time needed for execution, approximate execution start time, minimum power to CPU, etc.) to each discovered nodes. When Analyzer and Producer section have received the packet, at the first time, it sends job information to event recorder to register on database; after that, if this request was acceptable according to existing resources on node, then OCBR executer will start to carry out case-based reasoning algorithm. When decision tree was created and training set was selected based on new sample (future job) place on DC branches to produce results by

OCBR, at the end, Analyzer and Producer section will produced some useful rules based on DC branch and will compute some information, that is mentioned in below, along with obtained prediction for creating XML document to send for scheduler.

For example, some rules are as following.

Ratio of Fault for desired node= 7%

If day=Monday and time>8 and time<18 then fault=46/100; *{node have the most fault in previous days (46 % of 0.07).}*
If Tj<800s and CPU_Idle>80% and time>15 and time<22 then success=98/100; *{node have only 2% fault}*

Moreover, each node has computed the fowling measures in order to send to scheduler.

- Average Hit Ratio (HR): This attribute represents an average rate of success in all previous times.
- Hit Ratio for the last twenty jobs (RT).
- Hit Ratio for this time-period on previous days (TP) based-on a estimation on coming job execution. For example, how many jobs have been executed from 1.30 AM to 2.00 AM? This is important because some nodes in grid have failed much in specific time-period.

The following observations are considered:

- *$Count(SJ_i)$: returned the number of successfully finished jobs on node$_i$*
- *$Count(FJ_i)$: returned the number of failed jobs on node$_i$*
- *$Count(ST_i)$: returned the number of successfully finished jobs in the last 20 submitted jobs on node$_i$*
- *$Count(APJ_i)$: returned the number of all jobs submitted at this time, on the previous days on node$_i$*
- *$Count(NSP_i)$: returned the number of all jobs successfully finished at this time, on the previous days on node$_i$*

- $HR_i = \frac{Count(SJ_i)}{Count(SJ_i)+Count(FJ_i)}$
- $RT_i = Count(ST_i)/20$
- $TP_i = \frac{(Count(NSPi)*(Count(SJi)+Count(FJ_i)))}{Count(APJ_i)*(Count(SJi)+Count(FJ_i))-Count(APJ_i)^2+1}$
- $IDLE_i = \frac{(CPU\ IDLE_i)}{100}$
- $RAM_i = \frac{(Free\ Available\ phisical\ memory\ )_i}{SiZE\ of\ RAM_i}$

Then, this above computed result along with produced rules will send to scheduler in a XML document such as following code.

```
<result>
  <ID>54</ID>
  <HR>0.93</HR>
  <RT>0.95</RT>
  <TP>0.92</TP>
  <IDLE>0.78</IDLE>
  <RAM>0.7</RAM>

  ...
</result>
...
```

On the other hand, when Resource Analyzer take the above produced result, it will compute the below formula foe PR_i according to weight allocation to each parameter; of course, this weight identified after several experience.

$$PR_i = \frac{\begin{pmatrix} 0.5 * HR_i + 0.4 * RT_i + 0.8 * TP_i + \\ 0.4 * IDLE_i + 0.2 * RAM_i \end{pmatrix} * count(SJ_i)}{count(FJ_i)}$$

PR_i show a status of node i^{th} in previous history. In the similar conditions for several nodes, this computed result could be determinant. For example, we have three nodes that their conditions almost are same but PR_i for each node are different. How much each node's PR_i is high then its priority also is better than others. For the nonce, these attributes have considered in fuzzy behavior for reason to grant a proper weight to each them

5. PERFORMANCE EVOLUTION

To evaluate proposed scheduler technique, we simulate a grid environment. Although we had previously provided 64 nodes in different places, this had been done for other purposes; anyway, these samples were sufficient for this research. Therefore, we used these samples in our simulator. In this simulator we considered 64 nodes. Each simulated node had its own local database. As we mentioned before, in these local database all events about job submission on the node have been saved. Simulated platform has been written with C++ codes. In this environment, there are two level schedulers; high-level scheduler and local scheduler. In this research, we have concentrated on local scheduler because each scheduler can act independently from other local schedulers. Every scheduler in desired site is scheduling nodes independently from other scheduler. Suppose that all 64 nodes are available in simulated grid and we want to select some nodes for our purposes. When we want to start execution, at first, all 64 nodes will execute OCBR algorithm and then will send obtained results to TXD database. Then scheduler's application compares all gathered results to select the best fault tolerance. These results are produced by scheduler according to priority in table1.

Table 1. Produced result by scheduler according to priority of nodes.

Node's Name	priority	prediction	PR_i	CPU-idle	Free RAM	Small job	Med job	Larg job	...
$Node_1$	0.89	0.91	24.1	0.54	0.02	VG	G	G	...
$Node_2$	0.87	0.95	29.8	0.77	0.04	VG	M	M	...
$Node_3$	0.84	0.82	21.9	0.97	0.05	G	M	M	...
$Node_4$	0.94	0.95	62.3	0.87	0.43	VG	VG	G	...
$Node_5$	0.95	0.96	75.6	0.98	0.39	EX	VG	VG	...
$Node_6$	0.92	0.95	56.4	0.46	0.38	VG	VG	G	...
$Node_7$	0.90	0.93	42.5	0.8	0.28	G	G	M	...
$Node_8$	0.78	0.80	20.85	0.65	1	G	M	VW	...
$Node_9$	0.92	0.91	48.9	0.8	0.28	VG	G	G	...
$Node_{10}$	0.83	0.89	34.85	0.65	1	G	G	M	...
...	...	...	...	...	...	...	...	...	...

EX= Excellent [95% to 100%];
G= Good [70% to 85%)
W= Weak [30% to 50%)

VG=Very Good [85% to 95%]
M= Medium [50% to 70%)
VW= Very Weak [0 to 30%)

Then we chose some nodes that had the best condition to execute an example job (memory need= 10.41and estimation time to execution= 850sec). After 14 experiments we reached objective results and then compared these results with another approach in [2]. The results have been represented in Figure 3.

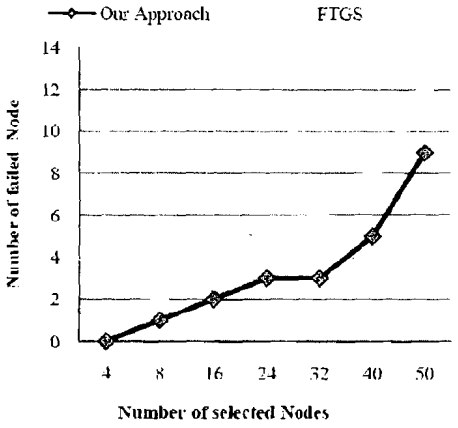


Figure 3: Evaluation the number of failure nodes in Our Approach with other Method

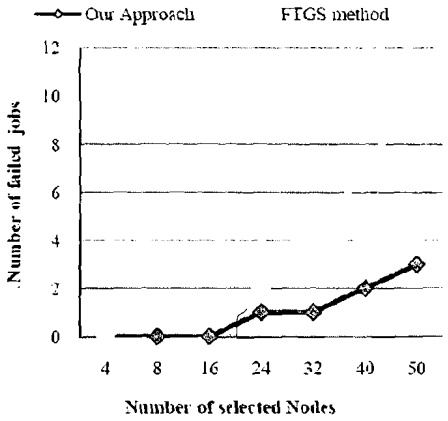


Figure 4: Evaluation the failure jobs in Our Approach with other Method

As you see, due to apply OCBR methods in resources scheduling, we obtained better results than FTGS method. Moreover, we compared successfully finished jobs in any way in these methods that results have been showed in figure 4.

It seems that, the New Approach tries to complete a job in one of the three existing ways. Therefore it has a good ability in successfully finishing job and it has a better fault tolerance feature. Also, in figure 5 we brought time execution in our approach and figure 6 show FTGS

method results. To reach this aim, we have done 3 experiments for small, medium, and large jobs. The results are mentioned below.

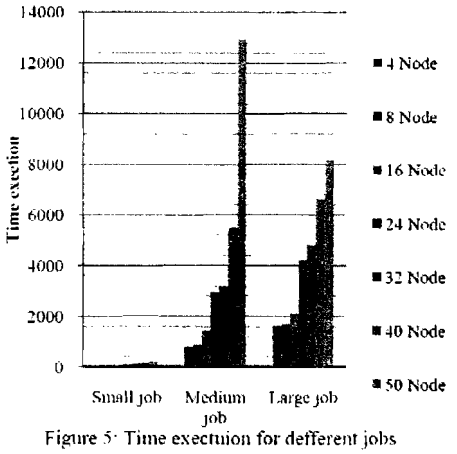


Figure 5: Time execution for defferent jobs on selected nodes in Our ApproachTitle

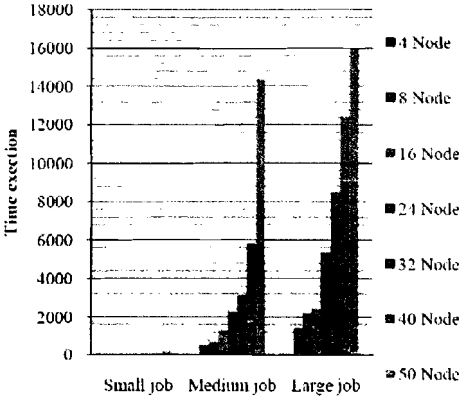


Figure 6: Time execution for defferent jobs on selected nodes in Other Method

6. CONCLUSION AND FUTURE WORKS

Since grid is a dynamic and active environment, job failures can occur for various reasons such as resource failure, network failure, RAM or Storage or CPU Problems, and software problems. In this paper, we proposed a predictive approach for resource management and scheduling with fault tolerance scheme for computational gird. This approach tries to use the CBR algorithm in order to find fault tolerance resources. The proposed grid-scheduling approach can select the best fault tolerance nodes and also detect a failure node and simply manage it by using one of the provided strategies such as multi-versioning, Reservation Queue and Replacement, and transferring job to the nearest neighbor. The obtained results show that the new approach may be very effective for adaptive grid scheduling due to reliability, fault tolerance, and then decrease of job completion time. As part of our future work, we are going to extend our approach by dynamic and intelligent component in scheduling phase.

ACKNOWLEDGEMENTS

This work was funded by Universiti Teknologi Malaysia (UTM). The authors would like to thanks to Mr. Ali Imani at UTM, Dr. Mr. Mohammadbager Karimi at IAUT (Islamic azad university- Tabriz branch, IRAN) for their helps in preparation of this paper.

REFERENCES

[1] A. Bouyer, B. Arasteh, A. Movaghar. "A new Hybrid Model using Case-Based Reasoning and Decision Tree Methods for improving Speedup and Accuracy". Iadis International Conference Applied Computing 2007, ISBN: 978-972-8924-30-0.

[2] B. Nazir, T. Khan. "Fault Tolerant Job Scheduling in Computational Grid". Proceedings in 2nd IEEE International Conference on Emerging Technologies, IEEE 708, November 2006.

[3] F. Favarim, J. Silva, F. Lau, C. Lung, M. Correia. "GRIDTS: A New Approach for Fault-Tolerant Scheduling in Grid Computing". Proceedings in Sixth IEEE International Symposium on Network Computing and Applications (NCA 2007)

[4] H. Bin Lenando. et al, "Applying Case-Based Reasoning (CBR) in Networked Appliances (NAs)". 2005

[5] I. Foster, C. Kesselman, S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", International J. Supercomputer Applications, 15(3), 2001.

[6] I. Foster. What is the Grid? - A Three Point Checklist. GRIDtoday,1(6), July 2002.

[7] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwel, and J. V. Reich. "The Open Grid Services Architecture, Version 1.5". Technical report, Global Grid Forum, July 24 2006.

[8] L. Burchard, C. A. F. De Rose, H.Ulrich Heiss, B. Linnert and J. Schneider. "VRM: A Failure-Aware Grid Resource Management System". Proc. of the 17th Intl: Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'05). IEEE.2005.

[9] R. Fernandes L. and F. Jos'e da Silva e Silva, "Fault Tolerance in a Mobile Agent Based Computational Grid." Proc. of the Sixth IEEE International Symposium on Cluster Computing and the Grid Workshops (CCGRIDW'06). 2006

[10] R. Medeiros, W. Cirne, F. Brasileiro and J. Sauve,.Faults in Grids: Why are they so bad and What can be done about it? in the proceedings of the Fourth Intl: Workshop on Grid Computing (GRID'03), 2003.

[11] R. Duan, R. Prodan, T. Fahringer. " Short Paper: Data Mining-based Fault Prediction and Detection on the Grid". IEEE, 2006.

[12] R. Koo and S. Toueg. Check pointing and rollback-recovery for distributed systems. IEEE Transactions on Software Engineering, 13(1):23–31, 1987.

[13] S. Baghavathi Priya, M. Prakash, K. K. Dhawan. "Fault Tolerance-Genetic Algorithm for Grid Task Scheduling using Check Point". Proceedings in IEEE the Sixth International Conference on Grid and Cooperative Computing (GCC 2007)

[14] T. Xie, X. Qin, A. Sung: SAREC: A Security-Aware Scheduling Strategy for Real-Time Application on Clusters," Proc. Int'l Conf. Parallel Processing (ICPP-2005), pp. 5-12, June 2005.