A Prediction-based Fault Tolerance on Grid Resources scheduling by using Optimized Case-based Reasoning

Asgarali Bouyer, Mohd Noor MD SAP
Faculty of computer science and information systems
University technology of Malaysia
{basgarali2@siswa.utm.my}, {Mohdnoor@fsksm.utm.my}

Abstract

Grid scheduling process is a main factor that affects system performance. If the grid scheduler is enabled to selecting proper resources and determining order of jobs in queue, each job is executed without missing their deadline and extra faults; and consequently, the response time of job is decreased. Therefore, using a better approach to resource scheduling to avoid fault is necessary. This paper presents a new approach on fault tolerance mechanisms for the resource scheduling on grid by using Rough Set analysis in a local fashion and Case-Based Reasoning technique on scheduler machine. This approach applies a specific structure in order to prepare fault tolerance between executer nodes to maintain system in a safe state with minimum data transferring. Certainly, this algorithm increases fault tolerant confidence therefore, performance of grid will be high. This method in a small scale is evaluated and result was acceptable.

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

1. Introduction

Grid computing [9] is a promising infrastructure to solve some problems that need to strong and heavy computation with very long time execution in a distributed fashion with low cost. In this environment, the computational 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. 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. 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, 1].

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 [2] 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. Moreover, we use Rough Set analysis to get some useful rules in order to handle by scheduler. Rough Set theory was introduced by Pawlak in 1982 [15]. RS, as a mathematical tool to deal with uncertainty and vagueness in data, provides us with a sound theoretical basis to determine the properties that define similarity [11].

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 rough set analysis and section 4 introduces the optimized CBR that is used in our method. Section 5 discusses the system design and implementation details of our Grid resource scheduling respectively. Section 6 describes experimental results and section 7 concludes the paper.

2. Related works

Nowadays, many researchers have investigated in grid computing to offer required methods for dependable task scheduling. The most objective in grid is failure-nodes problem in task scheduling. In the grid, it is possible that computational nodes are crashed and this problem is not deniable. In the past, many fault tolerance job scheduling techniques have been suggested for grid or cluster computing [16, 3, 18, 17, ref1, 7, and 14].

In previous work [3], we present a fault tolerant method for task scheduling based on OCBR technique in a local fashion. This method was one of the acceptable approaches. However, it is not suitable for the tasks with low or even medium completion time because OCBR take more time. In this paper, we develop the mentioned method by using Rough set theory to cover all tasks.

An extension of the scheduling algorithms of the GrADS tool has been presented in [19]. This is a Clustering-based Grid Resource Selection (CGRS) resource selection algorithm establishes on a new density-based grid resource-clustering algorithm and data mining method, which try to reduce the computational complexity, and providing single-site scheduling in case of invalidation of multisite resource selection. It handles multiple sites in resource selection policy. Resource selection algorithms can be classified into single-site (e.g. Nimrod/G [5]) and multi-site resource selection algorithms. This algorithm has an efficient policy to select resources from multi site, but it will act well only if optimal conditions and if there not exist enough resources in desired grid, it will encountered with some problems in making a better decision.

GRIDTS [7] 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 [14]. 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 [10, 4, 13, and 6] that we devolve them to readers.

3. Rough Set in Grid scheduling

Rough Sets Theory [15, 11] has often proved to be an excellent tool for the analysis of vagueness and uncertainty inherent in making decisions. It also proved to be very useful for analysis of decision problems concerning objects described in a data table.
by a set of condition attributes and by a set of decision attributes. Rough Sets can be applied on information represented in the form of a table called information system, Decision Tables or information tables.

We present some concepts of the rough set theory used in the process of generating equivalence classes in order to use in our approach for selecting the suitable resources for scheduling purpose.

Our purpose is to carry out resource selection for desired job’s condition in the Grid scheduling cycle. To do this issue we will use classification algorithm based on rough set theory. It takes the Nodes Information Data Table (NIDT) as input. In the context of rough set theory, Data are often presented as a table, columns of which are labeled by attributes, rows by objects of interest and entries of the table are attribute values. The output is the identified split point for DT and at the end; final output is classified data for applying by provided application.

The incoming data table as a whole can be treated as an information system of the form L =< U, A, V, f >. Here, U = {x₁, x₂, ...xₙ} is a nonempty set of objects (NIDT data) and n is the number of objects in NIDT data; A = C∪D, in which C={c₁, c₂, ...cₘ} (m is the number of conditional attributes) is a nonempty set of conditional attributes, and D is a finite set of decision attributes.

Also, V = ∪ₐ∈₄ Vₐ, where Vₐ is a domain (value) of the attribute a, and there is a map f: U×A→V is called the information function, such that f(x, a) ∈ Vₐ for every a∈A, x₁ ∈ U.

- **Indiscernibility Relation:** In a Information system L, for every subset of attributes B ⊆ A, a binary relation, denoted by IND (B), called the B -indiscernibility relation, is associated and defined as follows:

IND(B)={ (xₐ,xₜ)∈U×U: a∈B, f(xₐ,a)=f(xₜ,a) }

- **Reduction:** A Information system L, the reduction of condition attribute C means a nonempty subset R ⊆ C satisfied by the IND(R)= IND(C) condition.

Let X is the target Class; the lower approximation B(X) contains all the patterns or equivalence classes of B- indiscernibility relation that definitely belong to the class X and the upper approximation B(X) permits overlap. Since the upper approximation permits overlaps, each set of data points that are shared by a group of classes define indiscernible set. Thus, the ambiguity in assigning a pattern to a class is captured using the upper approximation.

- **Positive Regions and Dependency:** Let IND(B) denote the set of equivalence classes of U with respect to B (and B ⊆ C and A = C∪D). Also Let IND(D) denote the set of equivalence classes of U with respect to D. The positive region of B in IND(D) is Defined as

POS_B(D) = ∪ (B(X): X ∈ IND(D)).

That is, POS_B(D) includes all objects that can be sorted into classes of IND(D) without error, based exclusively on the classification information in IND(B).

Furthermore, we say that the set of attributes D depends in degree k on the subset R of C if:

k(R, D) = card(POS_B(D)) ÷ card(POS_C(D)).

Clearly, k(R, D) ≤ K(C, D).

Also, the following are equivalent:

k(R, D) = k(C, D), R is a reduction of C with respect to D,

POS_B(D) = POS_C(D).

Furthermore, since we have a numerical description of dependency, we can speak of near-perfect reducts (i.e. k(R, D) ≈ k(C, D)), so that we may discard even more attributes if they have little effect on the dependency.

In this approach, decision attribute will be selected based on user request (which parameter has received the highest score by user or applicant?).

There are a number of different algorithms suggested for data discretization, but here due to do a fast discretization, in our method, we use a manual discretization as following:

- **step1:** A= the average value for attribute a. ((Σ Vₐ)/n and n is the number of objects).
- **step2:** Min= minimum (Vₐ), Max= Maximum (Vₐ).
- **step3:** repeat step 3 for every Vₐ
  - if Vₐ ≥ (A+Max)/2 then Vₐ = 1
  - else if [(A-Min)/2 < Vₐ < (A+Max)/2] then Vₐ = 2
  - else Vₐ = 3

For finding the best Reduct, we also have considered a greedy algorithm. At the first, Relative Core must be defined. Relative core is defined by: CORE(C, D) = \{a∈ C: POS_C( D) ≠ POS_C-{a} (D)\}, where C and D denote two attribute sets. The set Core is called D-core of C, and denoted by POS_C(D).

After that, then we pass the core to the following algorithm:

R = CORE(C, D) and T = C - R.

Repeat step 2 While k(R, D) < k(C, D) - edge

Find attribute a ∈ T such that k(R ∪ {a}, D) is maximized.

Set R = R ∪ {a}; T = T - {a}.

Send to output R.

4. **OCBR algorithm**

This method applies CBR algorithm by using Decision Tree in order to select suitable sampling based on produced rules by Rough set analyzer. 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 [8] 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 Rough Set’s Rule classifier). In this approach, after produce suitable rule by each node, these rules are sent to coordinator (on scheduler machine). Then, OCBR-executer classifies the existing Rules (received from all nodes) based on similarity to new desired job.

At first, 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).

5. 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 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.

5.1. 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 rough set analysis in local database by desired node based-on this coming job. We use rough set analysis due to many reason: 1) rough set is faster than other techniques, and 2) rough set is enable to produce useful rules even though it work with small data set and also in large data set we can select exact rules. 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 database (TempDB).

Phase 2: coordinator will insert all received results from each node in a database. After that, OCBR-Executer will select useful rules and then nodes and then deliver the result to Resource analyzer(RA). After that, RA will analyze this results based-on fault tolerant criteria and some important job condition by executing several queries to get better and squarer decision. For instance, suppose that we need to select one node for our task and also OCBR-Executer select two nodes and deliver them to RA. Therefore, to get the best

Figure 1: An architecture for fault tolerance grid scheduling.
selection, RA considers some factor as an important factor. For example, one of these nodes has $\alpha$-rate fault tolerance for small task but for heavy task it node is not suitable whereas, other node is probably suitable for medium or heavy task with the same $\alpha$-rate fault tolerance. 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.

### 5.1.1 Purposed 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 technique [1] in while of executing job.

![Image](image-url)

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

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).

5.1.2 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.

5.2. 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 Rough Set Analyzer will start to get useful rules. Moreover, each node has computed the following 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:

- \( \text{Count}(SJ_i) \): returned the number of successfully finished jobs on node \( i \),
- \( \text{Count}(FJ_i) \): returned the number of failed jobs on node \( i \),
- \( \text{Count}(ST_i) \): returned the number of successfully finished jobs in the last 20 submitted jobs on node \( i \),
- \( \text{Count}(APJ) \): returned the number of all jobs submitted at this time, on the previous days on node \( i \),
- \( \text{Count}(NSP) \): returned the number of all jobs successfully finished at this time, on the previous days on node \( i \),

\[
HR_i = \frac{\text{Count}(SJ_i)}{\text{Count}(SJ_i) + \text{Count}(FJ_i)}
\]

\[
RT_i = \frac{\text{Count}(ST_i)}{20}
\]

\[
TP_i = \frac{\left( \text{Count}(NSP) \cdot (\text{Count}(SJ_i) + \text{Count}(FJ_i)) \right)}{\text{Count}(APJ) \cdot (\text{Count}(SJ_i) + \text{Count}(FJ_i)) - \text{Count}(APJ) \cdot 2 + 1}
\]
Then, this above computed result along with produced rules will send to scheduler in a XML document such as following code.

```xml
<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 for $PR_i$, according to weight allocation to each parameter; of course, this weight identified after several experience.

$$PR_i = \left( \frac{0.5 \cdot HR_i + 0.4 \cdot RT_i + 0.8 \cdot TP_i}{0.4 \cdot IDLE_i + 0.2 \cdot RAM_i} \right) \cdot \frac{\text{count}(S)}{\text{count}(F_i)}$$

$PR_i$ 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.

### 6. 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 TempDB 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 table I.

<table>
<thead>
<tr>
<th>Node's Name</th>
<th>Priority Prediction</th>
<th>PR1</th>
<th>CPU idle</th>
<th>Free RAM</th>
<th>Small job</th>
<th>Med job</th>
<th>Large job</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node1</td>
<td>0.89</td>
<td>0.91</td>
<td>24.1</td>
<td>0.54</td>
<td>0.02</td>
<td>VG</td>
<td>G</td>
</tr>
<tr>
<td>Node2</td>
<td>0.87</td>
<td>0.95</td>
<td>29.8</td>
<td>0.77</td>
<td>0.04</td>
<td>VG</td>
<td>M</td>
</tr>
<tr>
<td>Node3</td>
<td>0.84</td>
<td>0.82</td>
<td>21.9</td>
<td>0.97</td>
<td>0.05</td>
<td>G</td>
<td>M</td>
</tr>
<tr>
<td>Node4</td>
<td>0.94</td>
<td>0.95</td>
<td>62.3</td>
<td>0.87</td>
<td>0.43</td>
<td>VG</td>
<td>G</td>
</tr>
<tr>
<td>Node5</td>
<td>0.95</td>
<td>0.96</td>
<td>75.6</td>
<td>0.98</td>
<td>0.39</td>
<td>EX</td>
<td>VG</td>
</tr>
<tr>
<td>Node6</td>
<td>0.92</td>
<td>0.95</td>
<td>56.4</td>
<td>0.46</td>
<td>0.38</td>
<td>VG</td>
<td>G</td>
</tr>
<tr>
<td>Node7</td>
<td>0.90</td>
<td>0.93</td>
<td>42.5</td>
<td>0.38</td>
<td>0.28</td>
<td>G</td>
<td>M</td>
</tr>
<tr>
<td>Node8</td>
<td>0.78</td>
<td>0.80</td>
<td>20.85</td>
<td>0.65</td>
<td>0.28</td>
<td>G</td>
<td>M</td>
</tr>
<tr>
<td>Node9</td>
<td>0.92</td>
<td>0.91</td>
<td>48.9</td>
<td>0.38</td>
<td>0.28</td>
<td>VG</td>
<td>G</td>
</tr>
<tr>
<td>Node10</td>
<td>0.83</td>
<td>0.89</td>
<td>34.85</td>
<td>0.65</td>
<td>0.28</td>
<td>G</td>
<td>M</td>
</tr>
</tbody>
</table>

EX= Excellent [95% to 100%]; VG= Very Good [85% to 95%];
G= Good [70% to 85%];
M= Medium [50% to 70%];
W= Weak [30% to 50%];
VW= Very Weak [0 to 30%];

Then we chose some nodes that had the best condition to execute an example job (memory need= 10.41 and 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.

As you see, due to apply Rough Set and OCBR methods in resources scheduling, we obtained better results than our previous method [3] and FTGS method. Moreover, we compared 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 our previous work [3] method results. To reach this aim, we have done 3 experiments for small, medium, and large jobs. The results are mentioned below.
7. Conclusion and Future works

In Grid environments, job failures can occur for various reasons such as resource failure, network failure and etc due to dynamic nature of grid. This approach apply the Rough Set Analysis algorithm on grid nodes (provider nodes) and OCBR algorithm on scheduler machine to select proper nodes for desired job based on entered values by Grid user with respect to user’s criteria. 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 by our simulation indicate 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.

References


