computing@computingonline.net
www.computingonline.net

ISSN 1727-6209
International Journal of Computing

FAULT TOLERANCE IN GRIDS USING JOB REPLICATION

Mohammed Amoon [1), 2)]

[1)] Computer Science and Eng. Dept., Faculty of Electronic Eng., Menofia University, Egypt
[2)] Dept. of Computer Science, RCC, King Saud University, P. O. Box: 28095-11437 Riyadh, Saudi Arabia
mamoon@ksu.edu.sa, m_amoon74@yahoo.com

Abstract: *As grids consist of a large number of resources, fault tolerance forms an important aspect of the scheduling process. In this paper, we address the problem of scheduling user jobs in grids so that failures can be avoided in the presence of resources faults. We employ job replication as an effective mechanism to achieve efficient and fault-tolerant scheduling system. Most of the existing replication-based algorithms use a fixed number of replications for each job which consumes more grid resources. We first propose an algorithm to determine adaptively the number of job replicas according to the grid failure history. Then we propose an algorithm to schedule these replicas. The proposed algorithms have been evaluated through simulation and have shown better performance in terms of grid load, throughput and failure tendency.*

Keywords: *Job scheduling; Fault tolerant; Replication; Grid Computing.*

1. INTRODUCTION

Grid Computing enables organization of geographically distributed computing resources in different administrative domains into a single, unified system for solving large-scale applications in science, engineering and commerce. Resources can be computers, storage space, peripherals, software applications, and data. A middleware software layer should be implemented to provide the basic services for resource scheduling, monitoring security and so forth [1].

Since grid resources are highly heterogeneous and vary dynamically, either by fault or shutdown, more faults are likely to occur in the grid computing environments [2-4]. Faults are due to the inherently unreliable nature of the grid include hardware failures such as computer crash, link down, etc. and software failures such as extra load, programs removal, etc.

Reliable applications are designed in such a way that grid can automatically recover from failures without affecting the performance user needs according to his required Quality of Service (QoS). Grids should tolerate faults and continue to operate to complete its computations. Fault-tolerant is the property that enables grid to carry on its computations even on individual component's failure without terminating the entire computation. Due to the diverse nature of grid and large-scale applications on grid, fault-tolerant becomes a challenge on developing, deploying and running applications on grid environments [1], [5-7].

In general, handling of failures by scheduling strategies can occur either before or after the scheduling of the resources. The first approach is called Proactive and the second is called Post-active. Post-active approaches, using techniques of job monitoring, are relatively easier to implement. On the other hand, proactive approaches require more information about grid resources and works in a probabilistic fashion.

In proactive approaches, such as replication, the decisions of how to address possible failures in the grid are made before the job is executed. An effective proactive approach should provide a way, with all available information considered, to avoid any job from any possible failures. This potentially reduces the failure rates within grid, and also increases the capacity and throughput. This is unlike to post-active methods where re-submission of jobs typically leads to a decrease in throughput [8], [9].

All mechanisms proposed to deal with fault-tolerant issues in grids are classified into three categories. The first one is called space redundancy or job replication. In this category, the same job is replicated to be executed on multiple undependable resources to guard the job against a single point of failure. The second category is called time redundancy or checkpointing and rollback. In this

category, the state of a running job is saved to a stable storage. This state can be used later in case of any fault to resume execution of the job instead of restating it. The third category is called adaptive. It uses both job replication and checkpointing to achieve the fault-tolerant.

In this paper, the job replication category is considered in order to create a proactive fault-tolerant scheduling system. In this system, two algorithms are proposed. One algorithm is for determining the number of replicas for each job, namely, Adaptive Job Replication (AJR). The second is used for selecting the resources that execute these replicas, namely, Backup Resources Selection (BRS). We then compare the proposed algorithms with existing algorithms based on fixed number of replicas and others based on dynamic number of replicas.

2. RELATED WORK

A large number of research efforts have already been devoted to fault tolerance in the area of distributed computing. However, a little work has been done for fault tolerance in grid environments. Aspects that have been explored include the design and implementation of fault detection services, as well as the development of failure prediction, and recovery strategies. The recovery strategies are often implemented through job replication, checkpointing or combined between them.

Replication is a key mechanism for developing fault-tolerant and highly available grids. Replication is based on the assumption that the probability of a single resource failure is much higher than of a simultaneous failure of multiple resources. It avoids job recomputation by starting several copies of the same job on different resources. With redundant copies of a job, the grid can continue to provide a service in spite of failure of some grid resources carrying out job copies without affecting the performance.

Checkpointing is the ability to save the state of a running job to secondary storage so that it can later resume its execution from the time at which it was last stored. The purpose of checkpointing is to increase fault-tolerance and to speedup application execution on unreliable systems.

The work in this paper depends on using the job replication mechanism. In the proposed system, we need to determine the degree of over-provisioning or job replicas as small as possible in order to minimize the system overhead.

J. Abawajy [10] presented a distributed fault-tolerant scheduling (DFTS) algorithm that couples job scheduling with job replication. He assumed that grid is divided into sites and each site has a scheduling manager. Each scheduling manager acts as a backup for another scheduling manager. The algorithm uses fixed number of replicas for each job. Each job replica is scheduled to a different site to be executed. The number of replicas is specified by the user at the time of job submission.

K. Srinivasa, G. Siddesh and S. Cherian [11] proposed an adaptive replication middleware which depends on data replication at different sites of the grid. The middleware dispatches replicas to different nodes and enables data synchronization between multiple heterogeneous nodes in the grid. Data sources are synchronized by using TCP/IP transfer protocol.

M. Chetepen et al [12] provided some scheduling heuristics based on task replication and rescheduling of failed jobs. Their heuristics do not depend on particular grid architecture and they are suitable for scheduling any application with independent jobs. Scheduling decisions are based on dynamic information on the grid status and not on the information about the scheduled jobs.

In [8], C. Jiang and et al proposed a replication based fault tolerant algorithm which schedules jobs by matching the user security demand and resource trust level. The number of job replications changes adaptively with the security level of the grid environment.

3. OUR SCOPE

Since grid resources are highly heterogeneous and dynamic, more failures are likely to occur in grid environments. These failures affect the time needed to execute jobs and then degrade the performance of the grid. This is because if a resource is unavailable, the system will search for another suitable resource to execute the job. Thus, there is a need to minimize the effect of these failures on performance, when occurred.

Our scope is to use replication strategy to tolerate failures as it is able to replicate each job to be executed more than one time on different resources. Replication provides an efficient way to guarantee the completion of jobs according to the QoS required by the user. Most of the existing replication based algorithms uses a fixed number of replicas [10], [13]. This fixed number of replicas can lead to excessive utilization of resources and also to longer response times of jobs.

The main contribution in this work is to design a proactive fault-tolerant system using the job replication strategy and to evaluate its performance.

The design comprises proposing two algorithms. The first algorithm is for determining the number of job replicas adaptively, namely, Adaptive Job Replication (AJR). Adaptive means the number of replicas is not fixed for all the jobs submitted. The second algorithm is for selecting resources that will execute replicas, namely, Backup Resource Selection (BRS). The proposed algorithms are compared with the algorithms in [10], [12].

4. GRID SCHEDULING SYSTEM

Most of the existing grid scheduling systems assumes the same architecture and the same set of modules. These modules include: a User Interface (UI) through which users can submit their jobs to the grid, a Grid Scheduler (GS) assigns jobs received from users to grid resources, scheduling decisions taken by the GS are based upon information provided by the Resource Information Server (RIS), which collects the resource capability information, such as CPU capacities, memory size, *etc.* The scheduling system acts not only as an interface between users and grid resources but also provide reliable service to users. So, in fault-tolerant architectures, additional component called Fault Handler (FH) is included to handle failures in the system. Handling failures include fault detection and fault recovery.

In this paper, the assumed scheduling architecture is shown in Fig. 1. It consists of geographically dispersed resources managed by a single administration unit. Currently only a grid environment with a single centralized GS and RIS is considered. In the grid architecture considered, resources possess varying failure behavior. So, the considered architecture contains a fault handler that can deal with failures if happened.

The proposed scheme in this work considers the fault occurrence due to resource failure. For detection of such faults, grid scheduler allocates a job to a resource once the user QoS requirements are satisfied on cost, time or cost-time optimization. After allocating the job the resource, the scheduler expects the response of executing the job assigned to that resource within a certain time interval. This time interval is a function of the speed of that resource, communication latency between the scheduler and resource and queue length of the resource. If the resource could not give the result of job execution to the scheduler, it realizes that a fault has occurred and in next step information about that fault is maintained at resource information server (RIS), which helps in taking replication decision while allocating job to that resource in the next time.

5. PROPOSED SYSTEM

The proposed system tries to avoid the occurrence of failures through applying proactive scheduling. Also, the system tries to minimize the effect of the failure if occurred. The system achieves this through job replication. Replication means generating multiple copies of the same job and these copies will be dispatched to be executed on multiple backup resources in parallel at the same time. So, if one resource fails the job still has the chance to be executed without effective delays on another resource. When any replica of a job completes, all other replicas of the job are located and terminated, freeing up the backup resources. The system determines the number of job replicas according to the failure tendency of the resources assigned to the job in the scheduling step. Also, it determines backup resources that will execute these replicas according to the current load of the resources allocated to the job.

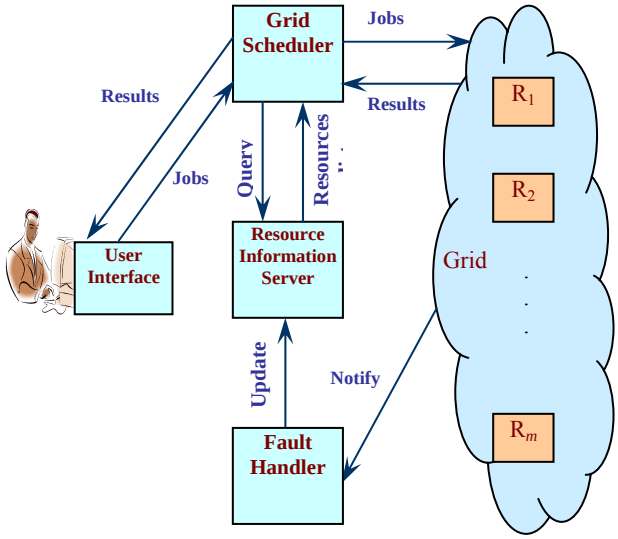

Fig. 1 – Architecture of the Proposed System

Thus, the main strategy of our proposed system is to minimize the effects of resources failures on the performance of the grid. To achieve this job replication mechanism is used.

The proposed system contains two main steps. These steps are allocation step and replication step. In the allocation step, the scheduler will select a set of suitable resources for executing the job. The selection of resources depends on the response time of the resources. The response time is the summation of the job transmission time from the scheduler to the resource on which the job will be executed, the job queuing time at the resource, the job execution time on the resource, and the transmission time of job's execution results from the recourse to the scheduler. Fig. 2 shows the operation of the proposed system.

```
For each job submitted by the user
{
Step 1(Allocation):
    Receives a job with QoS requirements from the portal;
    Requests a list of suitable resources the job from the RIS;
    Receives a list of suitable resources from RIS;
    Computes response time for each resource;
    Sorts resources in an ascending order according to the
response time;
    Determine the primary resource;
Step 2(Replication):
    Requests the FT for each resource from the Fault Handler;
    Requests the LH for each resource from the RIS;
    Computes number of replicas for the job;
    Determines the backup resources;
}
```

Fig. 2 – The proposed system's operation

The Job Scheduler will receive a job from the job queue along with the QoS required by the user. Then, it will ask the Resource Information Server for a list of suitable resources for executing the job according to the QoS requirements of the user. The Resource Information Server will reply with a list of resources and their expected response times for the job. The scheduler will sort this list according to the response time of each resource. The first resource in the sorted list is selected as the primary resource to execute the job.

As any resource in the grid, this primary resource may fail executing the job. So, in the replication step, the system will choose some resources from the list of allocated resources on which copies of the job will be executed. These resources are called backup resources.

Replicating a job can improve performance in one of two ways. First, by starting a job to run simultaneously on more than one resource, the job response time is determined by the earliest completion time among all replicas. As this can depend on unpredictable load and usage patterns, the replica can potentially improve the response time of the job. Second, replicated job executions can also help dealing with failure; then when one resource fails, the adverse effect on performance of the jobs it runs can be reduced if replicas complete without failure.

There are two challenges in this work. The first one is determining the optimal number of replicas in order to avoid increasing the response time of the job due to resources failure. The number of replicas should not be high in order to avoid grid overloading. The number of replicas or the backup resources is based on the failure tendency of the resources selected in the allocation step.

The second challenge is determining the backup resources that will execute job replicas. So, even in case of failure of a resource grid will be able to complete the job and to get results of the job's execution from an alternate resource. In this work, the selection of such resources depends on the current load of the resources selected in the allocation step.

6. ADAPTIVE JOB REPLICATION (AJR)

The AJR algorithm determines the number of job replicas. In the algorithm, the number of replicas is not fixed for all the jobs but it is dynamic. It is directly proportional to the tendency of the selected resources to fail. As the tendency of these resources to fail increases the number of replicas increases and vice versa. Thus, the number of replicas is based on failure history of the grid resources and this it will vary from job to job.

The failure tendency of a resource measures the expected degree of it to fail. This measurement depends on the history of the resource. Assume N_f is the number of times the resource has failed to complete jobs assigned to it and N_s is the number of times the resource has completed jobs successfully. Each time a resource is failed to complete a job the value of N_f is increased by 1 and the jobs assigned to that resource will be distributed to other suitable resources in the grid. Otherwise, the value of the N_s is increased by 1. The failure tendency FT_j of resource j is defined by:

$$FT_j = \frac{N_f}{N_s + N_f} \tag{1}$$

Assuming resources R_1, R_2, ..., R_n are allocated to the job j in the allocation step. The average failure tendency of these resources is:

$$FT_n = \frac{\sum_{j=1}^{n} FT_j}{n} \tag{2}$$

The number of job replicas, k, is determined to be proportional to the value of FT_n. At least, we must have one replica to be executed. So, the minimum number of replicas is 1. Also, the number of replicas should not exceed the number of suitable resources. So, the maximum number of replicas is equal to n.

Fig. 3 shows the algorithm used to determine the number of replicas for each job submitted. For each job, there will be at least one replica. The algorithm compares the value of FT_j for resource j with the value of FT_n starting from the first resource in the allocated list. If $FT_j >= FT_n$ then additional replica is added. The algorithm stops if $FT_j<FT_n$ for any resource j.

```
For each job i submitted by the user
{
  k = 0;
  j = first resource in the allocated list of resources for job i;
 Do
    {
    k ++;
    j = next resource;
    }
 While(FTj >= FTn);
}
```

Fig. 3 – AJR algorithm

7. BACKUP RESOURCES SELECTION (BRS)

After determining the number of job replicas, the backup resources will be selected. So, even in case of failure of a resource grid will be able to complete the job and to get results of the job's execution from an alternate resource. In this work, the selection of such resources depends on the current load of the resources selected in the allocation step.

The load of a resource is the size of computations done by the resource related to the resource speed. RIS is responsible for storing the load history of grid resources. The load history of a resource j is defined as:

$$LH_j = \frac{W_c}{S_j}, \tag{3}$$

where W_c is the total computation work, in instructions, done by the resource j and S_j is the speed of the resource measured in Million Instructions per Seconds (MIPS).

```
For each resource j in the allocated list;
{
    if(FTj > FTn)
        remove j from the list;
}
Sort resources according to LH in ascending order;
j = 1;
while(Replicas > 0)
        {
        Select j as a backup resource;
        Replicas--;
         j++;
        }
```

Fig. 4 – BRS algorithm

Backup resources will be selected from list of the allocated resources. The selection is based on the current load of these resources. Fig. 4 shows the BRS algorithm used to select the backup resources for a job. Firstly, the algorithm selects resources satisfying the condition: $FT_j < FT_n$. Then, the selected resources are sorted according to the current load of each resource. The sorting is done in ascending order. The first k resources are selected as the backup resources.

8. RESULTS AND ANALYSIS

Grid is a complex environment and it is difficult to build a grid on a real scale to validate and evaluate scheduling algorithms. Therefore, simulation is often used. There exist a number of well-known grid simulators, such as GridSim, SimGrid and NSGrid. However, all the above simulators do not support fault-tolerant and they have limited modeling for grid dynamics [12]. So, we developed a grid simulator using Java 6 to validate and evaluate our proposed fault-tolerant scheduling algorithms.

The simulator provides multiple services. These services include application composition and grid configuration through manual inputs from users, information services for resource discovery and fault detection, services for assigning application tasks to resources and managing their execution, and ability to inject grid with some faults. Also, it allows users to develop, validate and evaluate their own algorithms.

The simulator is portable, since it is available for Windows and Linux, and also other platforms. Also, it provides extensibility and flexibility to simulate the dynamic behavior of the grid. The simulator supports several types of dynamic system modifications such as alternating resource availability. The communication between the simulator's components is dome through using the message passing operations.

In this section, the performance of the proposed algorithms is compared against the performance of the replication-based algorithms in [10] and [12]. The comparison is performed within grid systems with varying load and reliability.

In the simulation experiments, application are modeled with different number of jobs ranges from 1000 to 5000. The size of each job is randomly selected from 1KB up to 10MB. The number of resources in the grid is around 1000 resources.

9. GRID LOAD

In this section, we show the performance on grid load. In Fig 5, we observe that the proposed AJR algorithm performs better than the DFTS algorithm [10]. This is because the DFTS algorithm assumed fixed number of replicas for each submitted job. This leads to extra unwanted load. On the other hand, the proposed AJR algorithm assumes a different number of replicas for each job according to the failure tendency of the resources. In general, the grid load of the two algorithms increases with

the increase in the number of jobs submitted. But, the AJR algorithm introduces a lower rate of increase than DFTS algorithm. This leads to a better degree of scalability.

Fig. 6 shows the Grid load comparison between the proposed AJR algorithm and the TR algorithm [12]. In TR, the number of replicas for each job depends on the number of free resources in the grid suitable. It is shown that the proposed AJR algorithm performs better than the TR algorithm [12]. This is because the proposed AJR algorithm considers the failure history of resources when determining the number of replicas for each job.

Fig. 5 – Comparison between the proposed AJR algorithm and DFTS algorithm

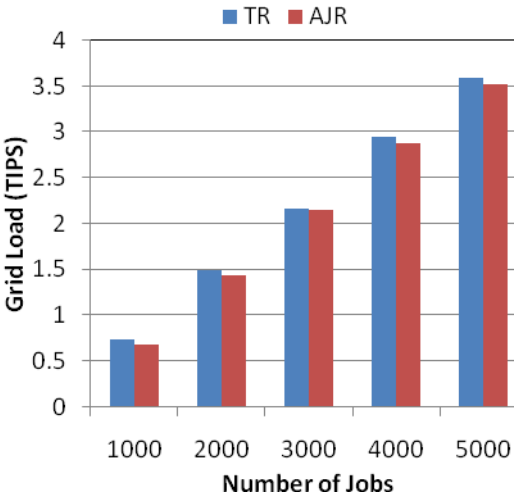

Fig. 6 – Comparison between the proposed AJR algorithm and TR algorithm

10. THROUGHPUT

Throughput is one of the most important standard metrics used to measure the performance of fault tolerant systems [9]. It is used to measure the ability of the grid to accommodate jobs. Throughput is defined as:

$$Throughput(n) = \frac{n}{T_n}, \qquad (4)$$

where n is the total number of jobs submitted and T_n is the total amount of time necessary to complete n jobs.

Fig. 7 and Fig. 8 show the throughput comparison of the proposed BRS algorithm with the ATR algorithm in [12] for different number of jobs submitted. The numbers of jobs are 1000, 2000, 3000, 4000 and 5000. The percentage of faults injected in the grid is 10% in Fig. 7 and 20% in Fig. 8.

Fig. 7 – Comparison between the proposed BRS algorithm and ATR algorithm with 10% injected faults

Fig. 8 – Comparison between the proposed BRS algorithm and ATR algorithm with 20% injected faults

It is shown from figures that the throughput of the BRS algorithm is better than the ATR algorithm for the whole range of job numbers. This is because in the BRS algorithm the failure rate and the current load of grid resources are taken into account when selecting the backup resources. On the other hand, The ATR algorithm selects free resources as the backup resources regardless of its tendency to fail. This leads to less faulty resources in the resulting schedule than the ATR algorithm.

11. CONCLUSION

In this paper, we proposed a fault tolerant scheduling system for grids that uses job replication mechanism. The system contains two main steps: allocation step and replication step. The main contribution of this paper is in the replication step. Two algorithms have been proposed and presented. The first determines the number of job replications and the second determines the backup resources on which replications will be executed. The performance of the two algorithms is evaluated under different conditions using metrics such as grid load and throughput. From results of experiments, it can be concluded that the proposed provides better performance.

12. REFERENCES

[1] S. Priya, M. Prakash and K. Dhawan, Fault tolerance-genetic algorithm for grid task scheduling using check point, *Proc. of the sixth International Conference on Grid and Cooperative Computing*, Urumchi, Xinjiang, China, August 16-18, 2007, pp. 676-680.

[2] S. S. Sathya and K. S. Babu, Survey of fault tolerant techniques for grid, Computer Science Review, (4) 2 (2010). pp. 101-120.

[3] Q. Zheng and B. Veeravalli, On the design of communication-aware fault-tolerant scheduling algorithms for precedence constrained tasks in grid computing systems with dedicated communication devices, *J. Parallel and Distributed Computing*, (69) (2009). pp. 282-294.

[4] H. Lee et al., A resource management system for fault tolerance in grid computing, *Proc. of International Conference on Computational Science and Engineering*, Vancouver, Canada, August 29-31, 2009, pp. 609-614.

[5] F. Khan, K. Qureshi and B. Nazir, Performance evolution of fault tolerance techniques in grid computing system, *J. Computing and Electrical Engineering*, (36) (2010). pp. 1110-1122.

[6] S. Hwang, C. Kesselman, A flexible framework for fault tolerance in the grid, *J. Grid Computing*, (1) (2003), pp. 251-272.

[7] H. Lee et al., A resource management and fault tolerance services in grid computing, *Journal of Parallel and Distributed Computing*, (65) (2005), pp. 1305-1317.

[8] B. Khoo and B. Veeravalli, Pro-active failure handling mechanisms for scheduling in grid computing environments, *J. Parallel and Distributed Computing*, (70) 3 (2010), pp. 189-200.

[9] M. Huda, H. Schmidt and I. Peake, An agent oriented proactive fault-tolerant framework for grid computing, *Proc. of International Conference on e-Science and Grid Computing*, Melbourne, Australia, Dec. 5-8, 2005, pp. 304-311.

[10] J. Abawajy, Fault-tolerant scheduling policy for grid computing systems, *Proc. of 18th IEEE International Parallel and Distributed Processing Symposium*, April 26-30, 2004.

[11] K. Srinivasa, G. Siddesh and S. Cherian, Fault-tolerant middleware for grid computing, *Proc. of 12th IEEE International Conference on High Performance Computing and Communications*, Melbourne, Australia, Sep. 1-3, 2010б pp. 635-640.

[12] M. Chtepen, B. Dhoedt, F. Cleays and P. Van-rolleghem, Evaluation of replication and rescheduling heuristics for gird systems with varying resource availability, *Proc. of 18th International Conference on Parallel and Distributed Computing Systems*, Anaheim, CA, USA, Nov. 13-15, 2006, pp. 622-627.

[13] S. Song, K. Hwang, and Y. Kwok, Risk-resilient heuristics and genetic algorithms for security-assured grid job scheduling, *IEEE Trans. Computers*, (55) 6 (2006), pp. 703-719.

M. Amoon has received the B.S. in 1996, M.S. in 2001, and Ph.D. in 2006 degrees from the computer science department, faculty of electronic engineering, Menofia University. He has been appointed as a Lecturer in the Department of Computer Science and Engineering at Menofia University. Now, he is assistant professor in the Department of Computer Science at King Saud University. His research interests include distributed computing, grid computing, Agent-based systems, Fault-Tolerant Computing and cloud computing.