J Supercomput (2012) 62:855–873
DOI 10.1007/s11227-012-0756-z

# Replication based fault tolerant job scheduling strategy for economy driven grid

**Babar Nazir · Kalim Qureshi · Paul Manuel**

Published online: 12 April 2012

**Abstract** In this paper, the problem of fault tolerance in grid computing is addressed and a novel adaptive task replication based fault tolerant job scheduling strategy for economy driven grid is proposed. The proposed strategy maintains fault history of the resources termed as resource fault index. Fault index entry for the resource is updated based on successful completion or failure of an assigned task by the grid resource. Grid Resource Broker then replicates the task (submitting the same task to different backup resources) with different intensity, based on vulnerability of resource towards faults suggested by resource fault index. Consequently, in case of possible fault at a resource the results of replicated task(s) on other backup resource(s) can be used. Hence, user job(s) can be completed within specified deadline and assigned budget, even on the event of faults at the grid resource(s).

Through extensive simulations, performance of the proposed strategy is evaluated and compared with the Time Optimization and Checkpointing based Strategy in an economy driven grid environment. The experimental results demonstrate that in the presence of faults, proposed fault tolerant strategy improves the number of tasks completed with varied deadline and fixed budget as well as number of tasks completed with varied budget and fixed deadline. Additionally, the proposed strategy used a smaller percentage of deadline time as compare to both Time Optimization

B. Nazir (✉)
Department of Computer Science, COMSATS Institute of Information Technology, 22060, Tobe Camp., Abbottabad, Pakistan
e-mail: babarnazir@gmail.com

K. Qureshi · P. Manuel
Department of Information Science, Kuwait University, Safat 13060, Kuwait

K. Qureshi
e-mail: kalimuddinqureshi@gmail.com

P. Manuel
e-mail: pauldmanuel@gmail.com

and Checkpointing based Strategy. Although the proposed strategy has a percentage of budget spent greater than that of Time Optimization Strategy and Checkpointing based Strategy, it is accepted as a proposed strategy in time optimization where the main objective is to maximize tasks completed within a given deadline. It can be concluded from the experiments that the proposed strategy shows improvement in satisfying the user QoS requirements. It can effectively schedule tasks and tolerate faults gracefully even in the presence of failures, but the costs are slightly higher in terms of budget consumption. Hence, the proposed fault tolerant strategy helps in sustaining user's faith in the grid, by enabling the grid to deliver reliable and consistent performance in the presence of faults.

**Keywords** Economy driven grid · Grid job scheduling · Grid resource management · Fault tolerance · Task replication · Grid computing

## 1 Introduction

The term *grid computing* was introduced in the early 1990s by Ian Foster and Carl Kesselman [1–4]. Grid computing uses idle computational power from different geographical places, by the runtime aggregation of these resources in the form of Virtual Organization [1–4], according to the needs of a job submitted by a grid user. It was intended to be done in the same pervasive fashion as we access the electric power grid [3, 4]. Lack of centralized controlled environment, predominant execution of long jobs, diversity of geographical distribution of resources from different administrative domains, and heterogeneous nature of grid resources in large scale grid increase the probability of failures exponentially. Consequently, in the absence of a fault tolerance mechanism, grid performance can considerably worsen, when compared with the traditional systems. Thus, the incorporation of fault tolerance related features in grid job scheduling policy is not an optional feature, but a necessity.

Economy Driven Grid model [5, 6] is a user-centric [7] approach. In an economy driven grid, a grid user submits a job to the grid resource broker by specifying job characteristics such as the number of tasks and their length, as well as Quality-of-Service (QoS) requirements, such as the computational strategy, processing time (deadline), and budget. In an economy driven grid, the job scheduling decisions are made dynamically at runtime, with the primary aim to satisfy the user's QoS requirements. Contrary to a conventional cost model, which often deals with software and hardware costs for running applications, the economy driven grid model primarily charges the grid user for services they consume, based on the value they derive from grid resources. As in the conventional market, a pricing model in a competitive economy driven grid is based on the demand of users and the supply of grid resources. An economy driven grid [8–10], on the one hand, offers profits for resource owners for contribution of their resources and, on the other hand, gives users a dynamic environment to optimize their gains given cost and time requirements.

The rapid growth of the Internet in the last 10 years was the first major facilitator of the renewed interest in fault tolerance and related techniques. We believe that the emergence of grid computing will further increase the importance of fault tolerance. Grid computing will impose a number of unique new conceptual and technical challenges to fault-tolerance researchers. Thus, the incorporation of fault tolerance related features in a grid job scheduling strategy should not be an optional feature, but a necessity. Fault tolerance becomes more critical when an economy based grid environment [11–13] is considered. Failing to meet the deadline and user's QoS will adversely affect user's faith in the grid, resulting in loss of business.

The motivation of this paper is to develop a fault tolerant grid scheduling strategy for economy based grid systems. We investigated the performance of efficient scheduling in the presence of faults so that the penalty paid by the resource provider is minimized [31]. This also enables the grid to uphold the faith of the user by not compromising its QoS requirements due to faults at grid resources. Some of the main contributions of the paper are:

1. We advocate the need for a fault tolerant job scheduling mechanism for an economy based grid environment. A fault tolerant grid-scheduling model for an economy based grid is proposed. This paper modifies the model for time optimization strategy presented in [8] and adds fault tolerant features.
2. The paper presents an adaptive task replication based fault tolerant job scheduling strategy for an economy based grid. It addresses the problem of fault tolerant job scheduling for an economy based grid. The proposed strategy uses an adaptive heuristic of task replication, enabling the grid to complete grid jobs within specified deadline and allotted budget. It is done by using the result of one of task replicas in case of possible fault occurrence at grid resource(s).
3. To simulate the proposed strategy, we enhanced the GridSim Toolkit-4.0 to exhibit fault tolerant related behavior. The paper defines the interaction protocol for communication between the core GridSim entities and new entities introduced by this paper for fault tolerance.
4. Through extensive simulation, the paper compares the performance of the proposed strategy with the time optimization strategy in economy grid environments.

The rest of this paper is organized as follows. Section 2 briefly explains review of important research efforts for providing fault tolerance in grid computing. Section 3 elaborates the problem formulation. In Sect. 4, explanation of proposed interaction protocol is given. Section 5 elaborates the proposed adaptive fault tolerant job scheduling strategy. Section 6 discusses simulation environment, experimental setup, results and their importance. The final section includes conclusions and suggests future work.

## 2 Related work

Fault tolerance or graceful degradation is the property of a distributed computing system which distinguishes it from sequential computing. It enables distributed computation to carry on its computation even on individual component's failure without terminating the entire computation [20, 21]. Due to the diverse nature of grids and large scale applications on grids, fault tolerance becomes a challenge in developing, deploying, and running applications on the grid environment [22, 23]. A grid environment, due to scale of complexity and heterogeneous nature of a grid, is more vulnerable to failures than traditional distributed systems. Hence the fault tolerance techniques (FTTs) of traditional distributed systems are not enough to manage faults in a grid environment. Therefore, we require special FTTs that could work well in complex and heterogeneous grids. Consequently, over the years researchers have yielded considerable theoretical and practical knowledge of fault detection, handling, and recovery techniques [20, 24].

In the literature, research on fault tolerance in the grid environment can be divided into two main types: pro-active and post-active. A pro-active fault tolerance mechanism takes into account the failure of a grid resource before scheduling jobs on grid resources. On the other hand, a post-active mechanism considers and takes appropriate measures on job faults after the job failure. Most researchers apply the latter approach to deal with failures using different methods such as grid monitoring approach as mentioned in [32].

As far as fault detection in any resource of a grid is concerned, there are two main strategies: the pull model and the push model as described in [11]. In the pull model, different grid components are responsible for sending periodic signals to a fault detector. In the absence of any such signal from any grid component, the fault detector recognizes that failure has occurred at that grid component. It then implements appropriate measures dictated by the predefined fault tolerance mechanism. In the push model, it is the responsibility of the fault detection component to send periodic signals to the different grid components. Furthermore, the fault detection component is responsible for detecting and processing the faults.

Task-level techniques refer to recovery techniques that are applied at the task level to mask the effect of faults irrespective of fault types [22, 25]. Task level FTTs [24] include the following:

1. *Retry*—The retry technique for fault tolerance [24] is the simplest technique being used. After a failure, it retries the task on the same grid resource regardless of the cause of failure up to some threshold value with the hope that there will be no failure in successive attempts.
2. *Alternate Resource*—The alternate resource technique works just like the retry technique except it retries on an alternate resource rather than retrying on the same resource again and again [25, 27].
3. *Checkpoint*—The checkpoint technique [19, 28, 29] periodically saves the state of an application. On failure it moves the task to another resource and starts the execution from the last saved checkpoint.
4. *Replication*— The replication technique in fault tolerance [23, 30] runs different replicas of same task on different grid resources simultaneously hoping that at least one of them will complete successfully.

Because of the simplicity of implementation, retry and alternate resource techniques are being used most often [27] as compared to replication and check-pointing [18, 26] techniques.

Workflow level FTTs [22] change the flow of execution on failure based on the knowledge of task execution context. Workflow level FTTs [24] are classified as follows:

1. *Alternate Task*—The alternate task technique is similar to the retry technique. The only difference is that it exchanges a task with different implementation of the same task with different execution characteristics on failure of the first one [25, 27].

2. *Redundancy*—The redundancy technique [30] requires different implementations of same task with different execution characteristics which run in parallel as opposed to task level replication technique where same tasks are replicated on different grid resources.
3. *User Defined Exception Handling*— In the user defined exception handling technique [24, 30], the user specifies a particular treatment to workflow of a task on failure.
4. *Rescue Workflow*—The rescue workflow technique [24] allows the workflow to continue even if the task fails until and unless it becomes impossible to move forward without catering the failed task.

In addition to the above mentioned basic fault tolerance techniques, there are several strategies using different heuristics to provide fault tolerance in grid computing. The following is a review of some important techniques.

In [12], an agent oriented pro-active fault tolerant grid framework is proposed. It divided the faults in a grid environment into six classes which include hardware faults, application and operating system faults, network faults, software faults, response faults, and timeout faults. Different software agents for different classes of faults are used to deal with faults in a proactive manner. These agents maintain information about different characteristics of the grid environment, such as memory consumption at resources, hardware conditions of resources, number of resources available, number of active processes at grid resources, condition of the network, and component Mean Time between Failures (MTBF). Based on this information and critical states collected by the agents, the agents enable the grid system to tolerate different types of faults gracefully.

In [13], mobile agents are used for providing fault tolerance, the mechanism is named MAG (Mobile Agents Technology for Grid Computing Environments), and also grid middleware is proposed. Here fault tolerant components are developed as mobile agents to provide fault tolerance in a grid environment. Mobile agents form a multi-agent society which provides fault tolerance for grid components.

In [14], when a grid node is recognized as faulty, all the jobs which are assigned to this faulty node are remapped to some other grid node. Moreover, no new jobs will be assigned for a specific period based on the Mean Time to Repair (MTTR) of the node.

A review of the references [12–14] and [22–30] reveals that grid environments are more failure-prone than general distributed systems. Fault tolerant measures in a grid environment are different from those of general distributed systems. Although there is some work done on fault tolerance in traditional and to an extent in a grid system, still, according to the literature, limited work is done on fault tolerance in an economy based grid environment. Thus, there is a need of new fault tolerance scheduling mechanism for the economy based grid environment.

## 3 Problem formulation

Grid jobs are executed by an economy based grid as follows:

1. Grid users submit their jobs to the grid resource broker (GRB) by specifying their QoS requirements, i.e., the deadline by which users want their jobs to be executed and the budget which users have for the completion of jobs.
2. The Grid Resource Broker schedules user jobs on the best available resource by optimizing time.
3. The result of the job is submitted to the user upon successful completion.

Such an economy based environment has the following two major limitations:

1. If a fault occurs at a grid resource, present heuristics merely reschedule the job on another resource, which eventually results in failing to satisfy the user's QoS requirements, i.e., budget and deadline. The reason is simple—the job is re-executed and hence consumes more budget and time.
2. In the economy based grid environments, there are resources that fulfill the criteria of deadline and budget constraints (QoS requirements), but they have a tendency towards faults. In such a scenario, the grid resource broker goes ahead to select the same resource for the mere reason that the grid resource promises to meet QoS requirements of the grid jobs. It eventually results in compromising the user's QoS parameters in order to complete the task.

In this paper, in order to address the first problem, a task replication heuristic is used. It enables the system to use a task replica successfully received from some other resource, when a failure occurs at any resource. Consequently, an economy based grid is able to tolerate faults gracefully.

In order to address the second problem, we make our replication strategy adaptive by maintaining a fault index. The fault index is maintained by taking into consideration the fault occurrence history information of the grid resource. In this way, the proposed strategy is able to introduce task replication mostly when it is necessary. Simulation experiments show that the proposed strategy is able to tolerate faults gracefully by taking appropriate measures according to resource vulnerability towards faults.

## 4 Adaptive task replication based fault tolerant job scheduling strategy for economy driven grid

In this section, the proposed system model and the proposed scheduling strategy are explained in detail (see Fig. 1). It explains how the proposed strategy works and enables the system to tolerate fault gracefully.

### 4.1 Grid user

A grid user submits a job to the grid by specifying job characteristics such as the number of gridlets and its length, Quality-of-Service (QoS) requirements such as the computational strategy, processing time (deadline), and budget. Here a gridlet [5] is a package that contains all the information related to the job and its execution management details, such as job length expressed in Million of Instruction Per Second (MIPS), disk Input/Output operations, the size of input and output files, and the job originator.

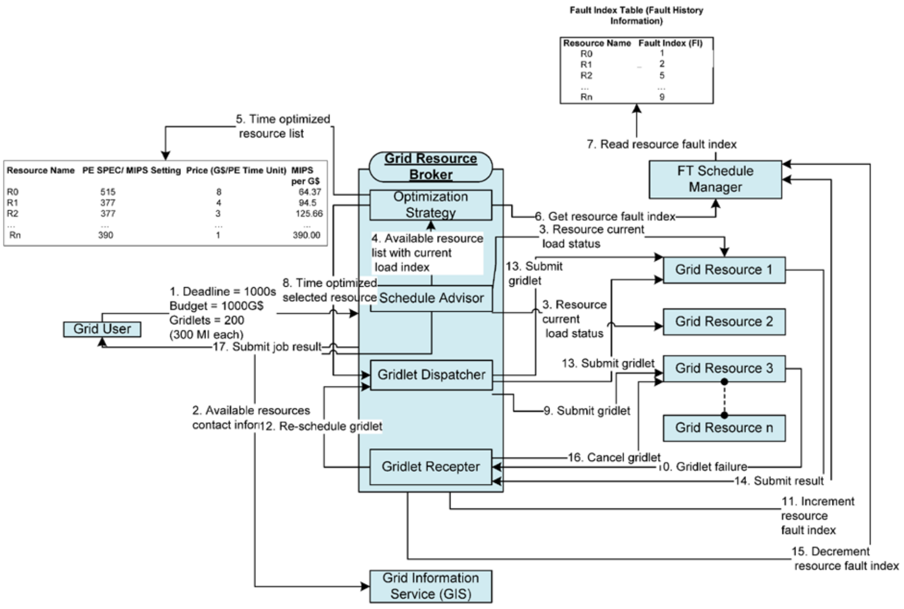
Fault Index Table (Fault History Information)
Resource Name Fault Index (FI)
R0 1
R1 2
R2 5
Rn 9
5. Time optimized resource list
Resource Name PE SPEC/ MIPS Setting Price (G$/PE Time Unit) MIPS per G$
R0 515 8 64.37
R1 377 4 94.5
R2 377 3 125.66
Rn 390 1 390.00
7. Read resource fault index
Grid Resource Broker
Optimization Strategy
FT Schedule Manager
6. Get resource fault index
3. Resource current load status
4. Available resource list with current load index
13. Submit gridlet
Grid Resource 1
8. Time optimized selected resource
Schedule Advisor
1. Deadline = 1000s Budget = 1000G$ Gridlets = 200 (300 MI each)
Grid User
3. Resource current load status
Grid Resource 2
17. Submit job result
13. Submit gridlet
Grid Resource 3
Gridlet Dispatcher
9. Submit gridlet
2. Available resources contact info
12. Re-schedule gridlet
Grid Resource n
16. Cancel gridlet
Gridlet Recepter
10. Gridlet failure
14. Submit result
11. Increment resource fault index
15. Decrement resource fault index
Grid Information Service (GIS)

**Fig. 1** Interaction of different grid components during job scheduling

## 4.2 Grid information server (GIS)

GIS contains information about all the available grid nodes with their computing capacity and cost at which they offer their services to the grid users. All the grid nodes joining and leaving the grid are known to the GIS. Whenever a grid broker has jobs to execute, it consults the GIS to find the currently available grid resources. At one time, services provided by each resource are static, and a resource is given only to those jobs for which they have registered their services at the GIS. When a resource want to add another service(s) it wants to provide, it will first register this service with the GIS, only then jobs for new services are send to them. The resource broker takes the resource allocation decision based on the information it gets from the GIS.

## 4.3 Grid resources

Grid resources offer their services to the grid users by registering them with the GIS by specifying their capabilities such as the number of processors, cost of processing, speed of processing, internal process scheduling policy, for example, time-shared (round robin) or space-shared (FIFO) local load factor, and time zone. The GIS maintains this information of grid resources. These basic parameters help in determining execution time, the time required to transport input and output files between users and grid resources, and returning the processed gridlets back to the originator along with the results. A grid user submits a job to the grid resource broker by specifying job characteristics and Quality-of-Service (QoS) requirements. The broker then schedules the tasks according to the scheduling policy in consultation with GIS.

### 4.4 Fault tolerant schedule manager

If a grid resource is unable to complete a job in a given time, it means that a fault has occurred. Fault index of a grid node will illustrate its vulnerability towards failure. Our strategy maintains and constantly updates the fault index about all the available resources in the grid environment. Fault index of a resource is incremented every time, whenever it fails to complete an assigned task as per contract. It is decremented when it competes successfully. Fault Tolerant Schedule Manager (FTScheduleManager) maintains the Fault Index Table about a grid resource and increments/decrements the resource fault index based on the feedback of the grid broker about its performance. It also maintains a gridlet replication table which contains information about one or two backup resource(s), where the gridlet is being executed in the form of a gridlet replica. The entries of the gridlet replication table are updated by receiving messages from the schedule advisor, when gridlets and their replicas are being assigned to different resources.

**Algorithm A:** Gridlet Resource Selection algorithm

```
Input:   List of available grid resources eligible for a job execution
Output:  Grid resource(s) selected for the job execution.
Begin
 1. while counter <= number of available resources do
 2. if selected resource has fault index >= 0 AND fault index <=2 then
 3.    The gridlet is queued to the selected resource.
 4.    GOTO Step6
 5. else if selected resource has fault index >= 3 AND fault index <=5 then
 6.    Gridlet is queued to that resource only:
 7.          if there exists one backup grid resource having fault index >= 4 AND
 8.           fault index <=7 then
 9.              Queue the gridlet to the selected resource AND Queue one replica of
10.              gridlet to that one backup resource.
11.              GOTO Step 6.
12.          endif
13. endif
14. if selected resource has fault index >= 6 AND fault index <=8 then
15.    Gridlet is queued to that resource only:
16.          if there exist two backup grid resources having fault index >= 4 AND fault
17.           index <=7 then
18.              Queue the gridlet to the selected resource AND Queue two replicas of
19.              gridlet to those two backup resources.
20.              GOTO Step6.
21.          end if
22. endif
23. if selected resource has fault index > 8 then
24.    Remove this resource from the table of available resources list and label as
25.    unavailable resource (i.e., no job is assigned) to that node until:
26.          It passes Alive Time successfully. The Alive Time is the time interval for which a grid resource remained
             up and connected. It is measured by sending periodic beacon signals to the resource for the set time interval
27.          (Alive Time). After surviving Alive Time, the grid resource is then included into the available resource list.
28. endif
29. Add the gridlet to the unassigned job list and reapply the time optimization algorithm to this gridlet on available
    resources.
30. endwhile
End
```

### 4.5 Schedule advisor

Schedule Advisor is an integral unit of the Grid Resource Broker. When a broker receives a job from a user, the Schedule Advisor prepares a list of resources with the help of the GIS that can execute the gridlet satisfying the QoS requirements. The Schedule Advisor gets the fault index of the selected resources from the FTSchedule-Manager and finalizes a suitable resource depending on the value of the Fault Index (a higher fault index depicts a more failure-prone and hence less reliable resource). Algorithm A implements the activities of the Schedule Advisor.

### 4.6 Gridlet dispatcher

The Gridlet Dispatcher dispatches the gridlets from the queue one by one to the respective resources for which they are queued for execution.

### 4.7 Gridlet receptor

The Gridlet Receptor collects the result(s) of the gridlet execution from the grid resource where the gridlet is dispatched by the dispatcher. The following Algorithm B implements the activities of Schedule Receptor.

**Algorithm B:** Fault Index Updating Algorithm.

```
Input:      Execution status of the assigned gridlet from the grid resource.
Output:     Update the fault index of the grid resource and cancel the gridlet replica(s) if required
Begin
  1.  Receive the execution status of assigned gridlet(s) from the grid resource.
  2.  if the grid resource successfully completes the execution of the gridlet then
  3.   Gridlet Receptor sends a message to FTScheduleManager to decrement the fault
  4.   index of the resource that successfully completes the assigned gridlet.
  5.     Gridlet Receptor checks the replication table from FTSchedule Manager.
  6.          if the replica of that gridlet is also being dispatched to some
  7.           backup resource(s) then
  8.               Send message to the resource to stop the execution of that
  9.               gridlet.
 10.               GOTO Step 3.
 11.          endif
 12.  endif
 13.  if  the grid resource fails to complete the execution of the gridlet then
 14.       Gridlet Receptor sends a message to FTScheduleManger to increment the
 15.       fault index of the resource that fails to complete the assigned gridlet.
 16.       Gridlet Receptor checks the replication table from FTSchedule Manager
 17.           if the replica of the gridlet is already dispatched to some backup
 18.            Resource then
 19.                Wait for the result from the backup resource.
 20.                 GOTO Step 1
 21.          endif
 22.          if the replica of that gridlet is not dispatched to a backup resource. then
 23.               Add the job to the list of unassigned jobs and send it to Schedule
 24.               Advisor for rescheduling.
 25.               GOTO Step3.
 26.          end if
 27.  end if
End
```

## 5 Interaction protocol for communication in GridSim environment

To simulate the proposed strategy, this paper enhances the GridSim Toolkit-4.0 [16] to exhibit fault tolerance related behavior. In GridSim based simulations, the interaction between different GridSim entities takes place through events. These events can be synchronous (wait until a destination entity performs required action), or asynchronous (do not wait for a destination entity to perform required action). Events can be internal (event destined to be delivered to the same entity that generated it) or external (event generating and destination entities are different). In GridSim, external events could be synchronous or asynchronous, but internal events must be asynchronous. The source entity can generate any event to itself or any other entity. Furthermore, any entity can be a source/destination entity. All the basic GridSim entities like a user, broker, resource, information service, statistics, shutdown, and report writer raise external events for other entities or internal event to itself for the request/delivery of any service.

To incorporate the fault tolerance feature, new entities in the GridSim environment are introduced. The interaction protocol for communication between different GridSim entities and proposed entities in our proposed scheduling model (see Fig. 2) will be as follows:

1. When GridSim starts, the resource entities that form the simulated grid environment, for example, resource entity 1 and resource entity 2 (see Fig. 2), register themselves with the Grid Information Service (GIS) entity, by sending events.

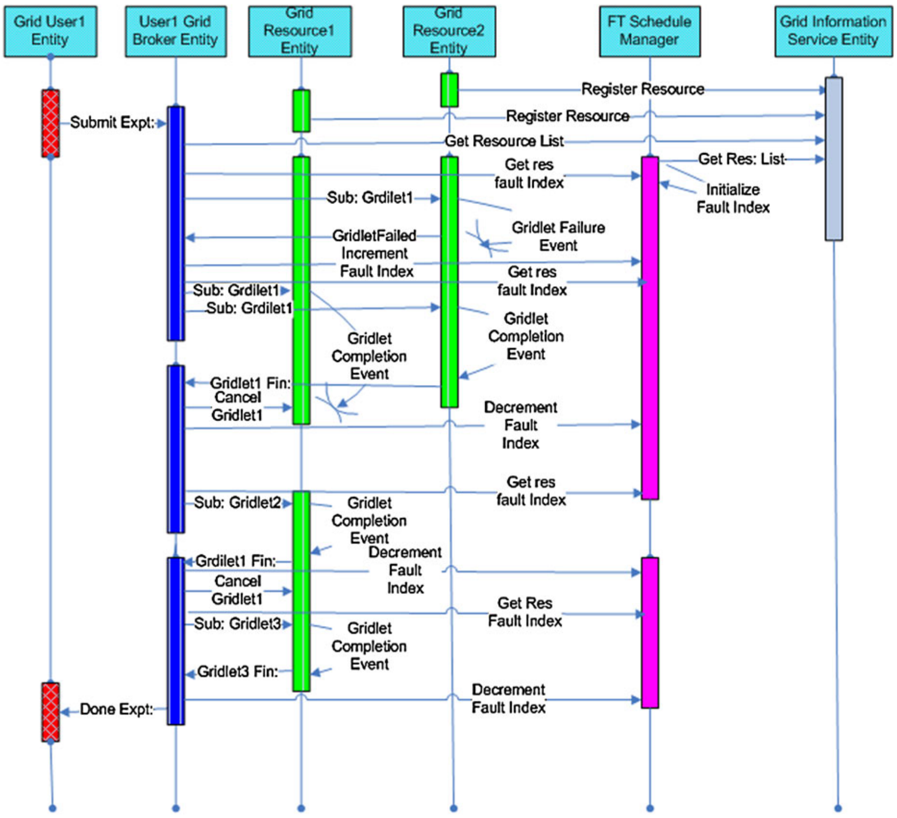

**Fig. 2** An event diagram for the Interaction Protocol for communication between GridSim entities

2. The Fault Tolerant Schedule Manager (FTScheduleManager) sends a synchronous event to the GIS entities to send the list of resource(s) available.
3. The GIS entity returns a list of registered resources to the FTScheduleManager. The FTScheduleManager initializes the resource fault occurrence history table that contains the fault index (suggesting resource vulnerability towards failures) of the resources available. As the simulation progresses, it will update that table (increment or decrement resource fault index) upon receiving an external event from the broker (explained below).
4. The broker sends a synchronous event for resource discovery to the GIS entity. The GIS entity returns a list of registered resources and their contact details.
5. After receiving the available resource list event from GIS, the broker entity finds the current work load condition of the resources. It sends events to resource entities requesting to send their resource configuration and properties.
6. Resource entities respond to this event by sending their dynamic information that includes resources cost, capability, availability, load, and other configuration parameters.
7. The broker entity selects the resource(s) for the execution of that gridlet, using time optimization scheduling strategy.
8. The broker entity sends synchronous events to the FTScheduleManager to get the resource fault index list.
9. The broker entity, after receiving the resource fault index list, dispatches gridlet(s) to resource entities. Furthermore, if needed (i.e., suggested by the fault index of selected resource(s)), the broker entity replicates the job to one or more resource(s) using Algorithm A. It dispatches a gridlet and its replica to one or more resource(s) according to Algorithm A.
10. At a resource entity when gridlet processing is finished, the resource entity sends a synchronous event of the completion of the gridlet to the gridlet receptor in a broker entity.
11. The gridlet receptor in a broker entity, on receiving the event from the resource entity, checks the event and does one of the following steps:
    (a) The CPManager entity, on the receipt of a checkpoint completion event from the resource entity, sends a gridlet completion event to the broker entity.
    (b) The CPManager entity, on the receipt of a gridlet failure event from a resource entity, sends an asynchronous gridlet failure event to the broker entity.
12. The broker entity, on a receipt of an event form the CPManager, does one of the following steps using Algorithm B:
    (a) The gridlet receptor in a broker entity, on a receipt of gridlet completion event from the resource entity,
        (i) Sends an asynchronous event to the FTScheduleManager entity to decrement resource fault index which completed the assigned gridlet.
        (ii) Sends an asynchronous event to the FTScheduleManager entity to check the replication table. It checks whether any of the replicas of this gridlet is assigned to any other resource(s). If yes, the dispatcher sends a synchronous event to that resource entity to stop/cancel execution of that job and decrement resource fault index which completed the assigned gridlet.
    (b) The broker entity, on a receipt of a gridlet failure event from a resource entity, does the following steps using Algorithm B:

(i) Sends an asynchronous event to the FTScheduleManager entity to increment resource fault index which failed to complete the assigned gridlet.

(ii) Sends an asynchronous event to the FTScheduleManager entity to check the replication table. It checks whether any of the gridlet replicas of this gridlet is assigned to any other resource(s).

1. If yes, it waits for the result of that replica from the resource entity to which it was assigned.
2. If no, it sends a synchronous event to the schedule advisor of the broker entity to reschedule the gridlet (see Fig. 2).

## 6 Experiment results and discussion

To evaluate the performance of proposed strategy, GridSim toolkit-4.0 [16] is used. In GridSim, time units are simulation time units which are modeled after the base value of SPECCPU (INT) 2000 benchmark ratings published in [17]. The budget is expressed in terms of Grid Dollar (G$) which is an artificial currency used in GridSim.

### 6.1 Simulation setup

A detailed description of the simulation environment in terms of the characteristics of resources simulated and specification of job submitted by a user is as below.

#### *6.1.1 Resource modeling*

For the simulation of the proposed strategy, GridSim Toolkit-4.0 [15] is used and modified to support fault tolerance related features. A number of space-shared resources are used with characteristics, cost and capability as they are used by World Wide Grid (WWG test bed) for economic based scheduling in [20] with modifications, i.e., all the resources used are made space shared and the processing element of a single resource is changed to 1. The simulated resources are obtained from three different sites taken from the WWG testbed with four resources from each site. To emulate failure behavior in the resources, we make four sub-types at each site resource, i.e. 0 % (no failures), 10 %, 20 %, and 30 %, where the failure rate suggests percentage of the grid resources that fail to complete during the simulation.

#### *6.1.2 Application modeling*

In simulation, one user is simulated which submitted 200 gridlets In GridSim, the length of a gridlet is expressed in Million of Instructions (MI) and the size of input file/output files is in bytes. The length of gridlets varies between 1000–12000 Million of Instruction Per Second (MIPS) and the size of input/output file varies in the range of 500–700 bytes at random.

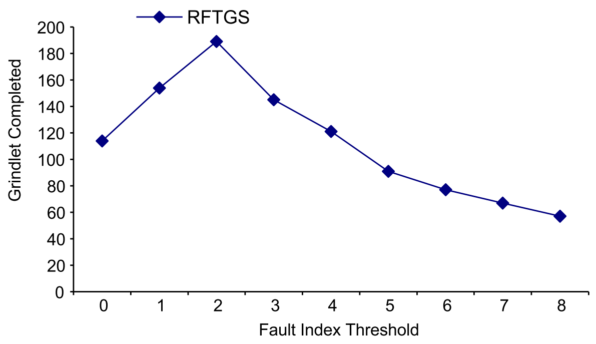

**Fig. 3** Number of gridlets completed for deadline (100 time units) and budget (21000 G$)

### 6.1.3 *Choice of fault index threshold*

In this experiment, 200 gridlets are submitted with a deadline of 100 time units and budget of 21000 G$. The fault index threshold is varied from 0 to 9. The purpose of this experiment is to find the optimum threshold of fault index in RFTGS at which resources are able to complete the maximum of gridlets. It is observed that (see Fig. 3) the RFTGS Strategy will be able to complete the maximum number of gridlets when the fault index threshold is 2. Similarly, the fault index threshold values are selected for threshold values 3, 6, and 8.

## 6.2 Results and discussion

In all the four proceeding experiments, there is only one user who submits 200 gridlets. In the following experiments, the performance of Replication Based Fault Tolerant Job Scheduling (RFTGS) Strategy is evaluated against Deadline and Budget Constrained Time Optimization Strategy [5] and Checkpointing Fault Tolerant Grid Strategy (CFTGS) [29].

### 6.2.1 *Experiment I*

In this experiment, 200 gridlets are submitted with a fixed deadline of 100 time units. The budget is varied from 5000 G$ to 21000 G$. The objective of the experiment is to compare the number of gridlets completed in the three strategies. As depicted in Fig. 4, it is observed that the RFTGS Strategy maintains its supremacy over Time Optimization Strategy as well as CFTGS throughout the experiment in terms of the number of gridlets completed.

Throughout the experiment, the Time Optimization Strategy had fewer gridlets executed than RFTGS within a fixed deadline. It is because of the fact that the Time Optimization Strategy assigns gridlets to grid resources without considering the fault occurrence history. On failure, it merely reschedules the gridlet on some other resource, which results in extra delay as the gridlet is executed on anther resource(s), ultimately decreasing the number of gridlets executed by the fixed deadline. RFTGS has more gridlets executed than CFTGS. Although CFTGS has the mechanism of periodically saving the state of an application and on failure it moves the task to another

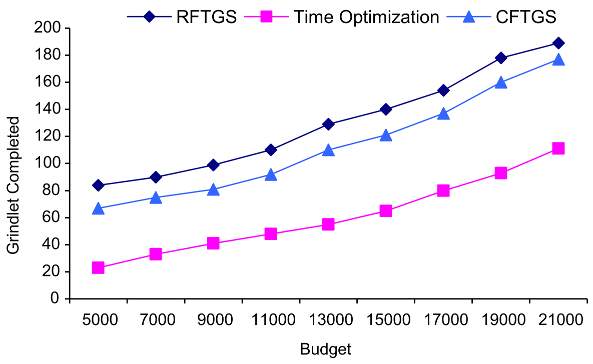

**Fig. 4** Number of gridlets completed for varied budget and fixed deadline of the three models

resource and starts the execution from the last saved checkpoint, it has checkpointing overhead as well as involves network delay in moving the checkpointed task and its result between the resource(s). At the same time, the RFTGS Strategy considers the reliability of a grid resource based on its fault index and replicates the task (using Algorithm A) on multiple grid resources only when it is required. Consequently, in case of failure at a grid resource, RFTGS Strategy will be able to get the results of the task from its replica, which saves time and resources to re-execute the task.

As both Time Optimization Strategy and CFTGS will have to reschedule the affected gridlet from scratch or from the last checkpoint results, it results in a delay in gridlet execution. For these reasons, the number of gridlets completed by the RFTGS Strategy is more than for both the Time Optimization Strategy and CFTGS.

### *6.2.2 Experiment II*

In this experiment, the budget is fixed at 5000 G$ and the deadline varies from 100 to 3700 time units. It is observed (see Fig. 5) that the number of gridlets completed by the RFTGS Strategy is more than for the Time Optimization Strategy and CFTGS.

RFTGS has more gridlets completed with lesser deadline than the Time Optimization Strategy within the fixed budget. The reason is that the Time Optimization Strategy has to re-execute the job in case of a failure, which results in extra budget consumption, whereas RFTGS at first has to consider the failure history (fault index) of the resource and accordingly take counter measures in terms of task replica to execute the gridlet. Thus, when a failure occurs it uses the result of a gridlet replica which decreases the budget consumption in the RFTGS.

The RFTGS has more gridlets completed than the CFTGS with smaller budget. It is because of the fact that in CFTGS extra budget is consumed in the event of a failure while re-executing the checkpointed task on some other resource. While in RFTGS, by virtue of a gridlet replica, there is no need to re-schedule the job, ultimately resulting in smaller budgets.

### *6.2.3 Experiment III*

In this experiment, the deadline is fixed at 100 time units and the budget varies from 5000 G$ to 25000 G$. We measure the percentage of deadline time utilized for dif-

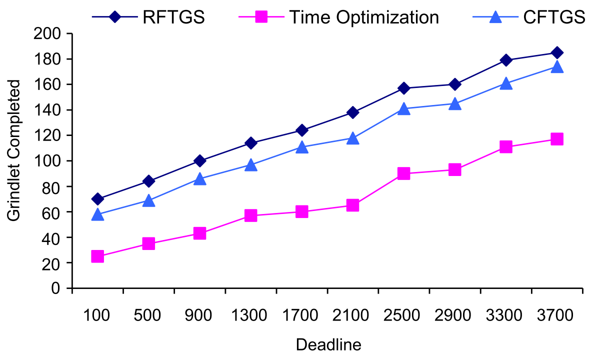

**Fig. 5** Number of gridlets completed for varied deadline and fixed budget of the three models

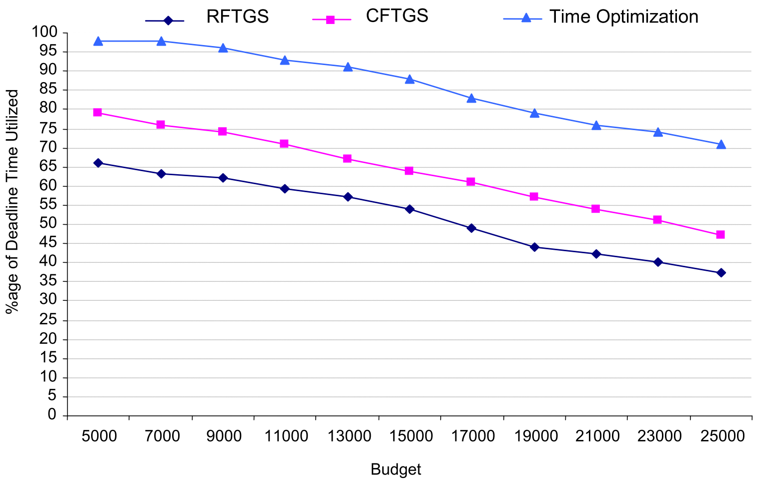

**Fig. 6** Percentage of deadline time utilized in the three models

ferent budgets. It is observed (see Fig. 6) that the RFTGS Strategy's percentage of deadline time utilization is less than for the Time Optimization Strategy and CFTGS.

The percentage of deadline time consumed by RFTGS is less than by the Time Optimization Strategy because in the Time Optimization Strategy a good amount of deadline time is consumed in re-scheduling the gridlet in case of a failure, whereas RFTGS maintains the history of fault occurrence at the grid resource in terms of fault index and schedules the gridlet on a resource proactively dealing with the possibility of gridlet failure suggested by fault index. Thus, the event of re-scheduling the gridlet from scratch which consumes most of deadline time is minimized in RFTGS. Consequently, the RFTGS has a smaller percentage of deadline time consumed than the Time Optimization Strategy.

The RFTGS has a smaller percentage of deadline time utilized than the CFTGS. It is because of the fact that CFTGS in case of a gridlet failure consumes most of its deadline time in re-scheduling the checkpointed gridlet on some other resource. On the other hand, RFTGS uses the gridlet replication heuristic to deal with a gridlet fail-

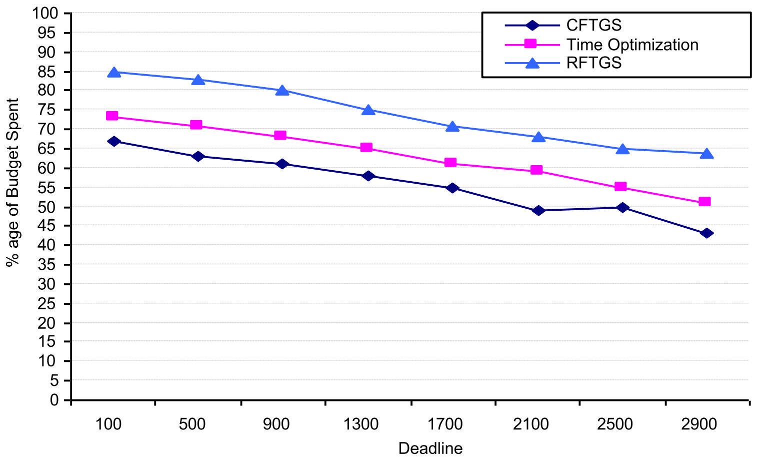

**Fig. 7** Percentage of budget spent in the three model

ure, hence avoiding the gridlet re-scheduling event. Hence, the RFTGS has a smaller percentage of deadline time consumed than the Time Optimization Strategy.

### *6.2.4 Experiment IV*

In experiment IV, the budget is 5000 G$ and the deadline varies from 100 to 2900 time units. Here we measure the percentage of budget spent for different deadline values. It is observed (see Fig. 7) that the RFTGS strategy's percentage of budget spent for different deadline time units is greater than that of the Time Optimization Strategy and CFTGS.

The RFTGS has the percentage of budget spent greater than the Time Optimization Strategy and CFTGS. The reason is that, to provide fault tolerance, the RFTGS uses the task replication on the backup resource(s). At times it results in redundant execution of gridlet(s), consequently causing extra budget utilization. The Time Optimization Strategy and CFTGS have no redundant executions of a gridlet. It ultimately contributes to minimizing budget consumption in both of these strategies. The over-consumption of budget in the RFTGS is acceptable in comparison to the advantage gained by the RFTGS in other parameters such as the number of gridlets completed for varied budget and fixed deadline, the number of gridlets completed for varied deadline and fixed budget, and the percentage of deadline time utilized. Additionally, as RFTGS is a time optimization strategy where the main objective is to maximize the number of gridlets executed within a given deadline, for such jobs it is more critical to complete a user job within a deadline rather than save budget.

### *6.2.5 Conclusion based on the experiments*

The experimental results demonstrated (see Table 1) that the RFTGS is able to maximize the number of gridlets completed for varied budget and fixed deadline, and number of gridlets completed for varied deadline and fixed budget. It suggests that the

**Table 1** Comparison of RFTGS strategy with time optimization strategy and CTGS strategy

| Performance comparison parameters | RFTGS strategy | Time optimization strategy | CFTGS strategy |
|---|---|---|---|
| Number of gridlets (21000 G$, 100 time units) | 95 % | 65 % | 84 % |
| Percentage of Deadline Time utilized (15000 G$, 100 time units) | 75 % | 96 % | 64 % |
| Percentage of Budget Utilized (1700 time units, 5000 G$) | 85 % | 74 % | 66 % |

RFTGS improved the overall execution time of the gridlets. Furthermore, it has minimized the percentage of deadline time utilized. Consequently, it can be concluded from the conducted experiments that the RFTGS effectively schedules gridlets and is able to tolerate faults gracefully even in the presence of failures, but costs are slightly higher in terms of budget consumption.

## 7 Conclusions

In this paper, the problem of fault tolerance in economy driven computational grid was considered and a novel adaptive task replication based job scheduling strategy for economy driven grid was proposed. The Proposed RFTGS Strategy included two algorithms to tolerate faults gracefully. The experiment results demonstrated that when compared with the CFTGS and Time Optimization Strategy, the RFTGS has improved the overall execution time of the gridlets with varied deadline and fixed budget as well as with varied budget and fixed deadline. Additionally, the RFTGS has outperformed the CFTGS and Time Optimization Strategy with respect to the percentage of deadline time utilized. Although the RFTGS has the percentage of budget spent greater than the Time Optimization Strategy and CFTGS, it is acceptable as the RFTGS is a time optimization strategy where the main objective is to maximize the number of gridlets executed within a given deadline, because for such jobs it is more critical to complete a user job within deadline rather than save budget. Thus, it is concluded that the RFTGS provides a suitable solution to fault tolerant scheduling in n economy driven grid environment and has shown improvement in satisfying the user QoS requirement. It enabled a grid to deliver reliable and consistent performance in the presence of failures.

## References

1. Foster I, Kesselman C, Tueke S (2001) The anatomy of the grid: enabling scalable virtual organizations. Int J Supercomp Appl 15(3)
2. Foster I, Kesselman C, Nick J, Tuecke S (2002) The physiology of the grid: an open grid services architecture for distributed systems integration. Technical Report, Open Grid Service Infrastructure WG, Global Grid Forum, June 2002
3. Foster I (2002) What is the grid? A three point checklist. In: GRIDToday, 20 July 2002
4. Foster I, Kesselman C (1999) The Grid: blueprint for a new computing infrastructure, Chap 2. Morgan Kaufman, San Mateo
5. Buyya R (2002) Economic-based distributed resource management and scheduling for grid computing. Ph.D. Thesis, Monash University, Melbourne, Australia

6. Buyya R, Abramson D, Venugopal S (2005) The grid economy. Proc IEEE 93(3):698–714. Special issue on grid computing. Parashar M, Lee C (eds)
7. Soysa M, Buyya R, Nath B (2006) GridEmail: economically regulated Internet-based interpersonal communications. In: Dai Y, Pan Y, Raje R (eds) Advanced parallel and distributed computing: evaluation, improvement and practice. Nova Science, New York, pp 279–295
8. Buyya R, Abramson D, Giddy J, Stockinger H (2002) Economic models for resource management and scheduling in grid computing. Concurr Comput 14(13–15):1507–1542
9. Buyya R, Murshed M, Abramson D, Venugopal S (2005) Scheduling parameter sweep applications on global grids: a deadline and budget constrained cost-time optimisation algorithm. Softw Pract Exp 35(5):491–512
10. Buyya R, Murshed M, Abramson D (2002) A deadline and budget constrained cost-time optimization algorithm for scheduling task farming applications on global grids. In: Proceedings of the 2002 international conference on parallel and distributed processing techniques and applications (PDPTA'02), 24–27 June 2002, Las Vegas, USA
11. Huda MT, Schmidt HW, Peake ID (2005) An agent oriented proactive fault-tolerant framework for grid computing. In: First international conference on e-science and grid computing (e-Science'05). IEEE Press, New York
12. Li Y, Lan Z (2006) Exploit failure prediction for adaptive fault-tolerance in cluster. In: Proceedings of the sixth IEEE international symposium on cluster computing and the grid (CCGRID'06)
13. Fernandes Lopes R, da Silva e Silva FJ (2006) Fault tolerance in a mobile agent based computational grid. In: Proc of the sixth IEEE international symposium on cluster computing and the grid workshops (CCGRIDW'06)
14. Burchard L-O, De Rose CAF, Heiss H-U, Linnert B, Schneider J (2005) A failure-aware grid resource management system. In: Proc of the 17th intl symposium on computer architecture and high performance computing (SBAC-PAD'05). IEEE Press, New York
15. Buyya R, Murshed M (2002) GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. Concurr Comput 14(13–15):1175–1220
16. Sulistio A, Yeo CS, Buyya R (2004) A taxonomy of computer-based simulations and its mapping to parallel and distributed systems simulation tools. Int J Softw Pract Exp 34(7):653–673
17. Singh G, Kesselman C, Deelman E (2007) A provisioning model and its comparison with best effort for performance-cost optimization in grids. In: Proceedings of the sixteenth IEEE international symposium on high-performance distributed computing (HPDC 2007), Monterey, California, USA, 25–29 June 2007
18. Nazir B, Qureshi K, Manuel P (2009) Adaptive checkpointing strategy to tolerate faults in economy based grid. J Supercomput 50(1):1–18
19. Nazir B, Khan T (2006) Fault tolerant job scheduling in computational grid. In: Proceedings of 2nd IEEE international conference on emerging technologies (ICET'06), Peshawar, Pakistan, pp 708–713, 13–14 November 2006
20. Stelling P, DeMatteis C, Foster I, Kesselman C, Lee C, Laszewski GV (1998) A fault detection service for wide area distributed computations. In: 7th IEEE international symposium on high performance distributed computing, p 268, Washington, DC, USA, July 1998. ISBN:0-8186-8579-4
21. Fault-tolerant system (2012) http://en.wikipedia.org/wiki/Fault-tolerant_system
22. Hwang S, Kesselman C (2003) A flexible framework for fault tolerance in the grid. J Grid Comput 1(3):251–272. doi:10.1023/B:GRID.0000035187.54694.75
23. Abawajy JH (2004) Fault tolerant scheduling policy for grid computing systems. In: 18th International parallel and distributed processing symposium (IPDPS'04), Santa Fe, New Mexico, 26–30 April 2004. IEEE Computer Society Press, Los Alamitos, pp 238–244
24. Yu J, Buyya R (2006) A taxonomy of workflow management systems for grid computing. J Grid Comput 3(3–4):171–200. doi:10.1007/s10723-005-9010-8
25. Gartner FC (1999) Fundamentals of fault-tolerant distributed computing in asynchronous environments. ACM Comput Surv 31(1):1–26
26. Anglano C, Canonico M (2005) Fault-tolerant scheduling for bag-of-tasks grid applications. In: Lecture notes in computer science, vol 3470/2005. Springer, Berlin, pp 630–639. doi:10.1007/b137919, ISBN:978-3-540-26918-2
27. Vanderster DC, Dimopoulos NJ, Sobie RJ (2007) Intelligent selection of fault tolerance techniques on the grid. In: Third IEEE international conference on e-science and grid computing. IEEE Computer Society, Washington. ISBN:0-7695-3064-8

28. Gioiosa R, Sancho JC, Jiang S, Petrini F (2005) Transparent incremental checkpointing at kernel level: a foundation for fault tolerance for parallel computers. In: Proceedings of the 2005 ACM/IEEE SC|05 conference (SC'05)
29. Jankowski G, Januszewski R, Mikolajczak R (2006) Grid checkpointing architecture—a revised proposal. In: CoreGRID TR-0036, 30 May 2006
30. Hwang S, Kesselman C (2003) Workflow grid: a flexible failure handling framework for the grid. In: 12th IEEE international symposium on high performance distributed computing (HPDC'03), Seattle, Washington, USA, 22–24 June 2003. IEEE CS Press, Los Alamitos
31. Yeo CS, Buyya R (2005) Service level agreement based allocation of cluster resources: handling penalty to enhance utility. In: Proceedings of the 7th IEEE international conference on cluster computing, cluster 2005, Boston, Massachusetts, USA, 27–30 September 2005. IEEE CS Press, Los Alamitos
32. Medeiros R, Cirne W, Brasileiro F, Sauvé J (2003) Faults in grids: why are they so bad and what can be done about it? In: Grid computing, 2003, Proceedings fourth international workshop, pp 18–24. ISBN:1-59593-414-6