J Supercomput (2011) 56: 106–128
DOI 10.1007/s11227-009-0345-y

A hybrid fault tolerance technique in grid computing system

Kalim Qureshi · Fiaz Gul Khan · Paul Manuel · Babar Nazir

Published online: 19 January 2010

Abstract In order to achieve high level of reliability and availability, the grid infrastructure should be a foolproof fault tolerant. Fault tolerance plays a key role in order to assert availability and reliability of a grid system. Since the failure of resources affects job execution fatally, fault tolerance service is essential to satisfy QoS requirement in grid computing.

In this paper we proposed two hybrid fault tolerance techniques (FTTs) that are called alternate task with checkpoint and alternate task with retry. These proposed hybrid FTTs inherit the good features and overcome the limitations of workflow level FTT and task level FTT. We evaluate the performance of our proposed FTTs under different experimental environments. Finally, we conclude that alternate task with checkpoint improves the reliability of a grid system more significantly than alternate task with retry.

Keywords Grid computing · Workflow level fault tolerance technique · Task level fault tolerance technique · Alternate task with checkpoint · Alternate task with retry

1 Introduction

Grid is an infrastructure that enables the integrated, collaborative use of high-end computers, networks, databases, and scientific instruments owned and managed by multiple organizations. Grid environment provides computing resources made available to the user as needed. The characteristics of grid infrastructure are cost, per-

K. Qureshi (✉) · P. Manuel
Information Science Dept., Kuwait University, Kuwait City, Kuwait
e-mail: qureshi@sci.kuniv.edu.kw

F.G. Khan · B. Nazir
COMSATS Institute of Information Technology, Abbottabad, Pakistan

formance, security, scalability, interoperability, reliability, flexibility, transparency, accuracy.

In short, the grid computing vision is to achieve a system which is automatically scalable, easy to use, secure, autonomic and fault tolerant. In modern computing world, corporate users emphasize on availability and reliability. In grid architecture with dozens of grid services, it is important for each of these services to be highly available since each service can affect most/all of other grid services. For each gridlet, reliability guarantees complete job execution, Guaranteed Delivery, Duplicate Elimination, and Message Ordering. In order to achieve high level of reliability and availability, the grid infrastructure should be a foolproof fault tolerant. Fault tolerance plays a key role in order to assert availability and reliability of a grid system. Since the failure of resources affects job execution fatally, fault tolerance service is essential to satisfy QoS requirement in grid computing. Some known domain-specific fault tolerance techniques such as checkpointing, master–worker and replication are briefly explained in the following.

Checkpointing scheme: A common method of ensuring the progress of a long running application is to take a checkpoint, i.e., to save its state on stable storage periodically. A checkpoint is an insurance policy against failures in the event of a failure; the application can be rolled back and restarted from its last checkpoint thereby bounding the amount of lost work to be recomputed. In case of any failure, Grid Checkpointing Architecture recovers the checkpointed application to the point where the last checkpoint was taken [1].

Snapshot scheme: The state of a distributed application consists of the instantaneous snapshot of the local state of processes and communication channels. However, in an asynchronous distributed system with no global clocks or shared memory, we can only devise algorithms to approximate this global state [2]. A snapshot is deemed consistent if it could have occurred during the execution of an application [2, 3]. To yield a consistent snapshot, an algorithm must ensure that all messages received by a process are recorded as having been sent.

Replication scheme: A fault-tolerant scheduling policy that loosely couples job scheduling with job replication scheme has a propose [4].

The fault tolerance or graceful degradation is the property of distributed computing system which distinguishes it from sequential computing. This property enables distributed system to carry on its computation even on individual component's failure without terminating the entire computation [5, 6]. Due to the diverse nature of grid and large-scale applications on Grid, fault tolerance becomes a challenge on developing, deploying and running applications on grid environments [7, 8].

Due to high level of complexity and heterogeneous nature of grid as compared to traditional computing systems, existing FTTs of traditional systems are not sufficient to manage faults in grid computing. Therefore we require special FTTs that could work well in complex and heterogeneous environments of grid. Consequently, over the years researchers have yielded a considerable body of theoretical and practical knowledge of fault detection, handling, and recovery techniques [6, 10].

Task level techniques refer to recovery techniques that are applied at the task level to mask the effect of faults irrespective of fault types [9, 12]. Task level FTTs [11] include the following:

1. *Retry*—Retry technique for fault tolerance [10] is the simplest technique being used. After a failure it retries the task on the same grid resource regardless the cause of failure up to some threshold value with the expectation 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 [12, 14].
3. *Checkpoint*—The checkpoint technique [15, 16] 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 [9, 17] runs different replicas of same task on different grid resources simultaneously expecting that at least one of them will complete successfully.

Because of the simplicity of implementation, retry and alternate resource techniques are being mostly used [14] as compared to replication and checkpointing [13] techniques. Workflow level FTTs [8] change the flow of execution on failure based on the knowledge of task execution context. Workflow level FTTs [10] are classified as follows:

1. *Alternate task*—is similar to retry technique. The only difference is that it exchanges a task with different implementation of same task with different execution characteristics on failure of the first one [12, 14].
2. *Redundancy*—The redundancy technique [17] 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 user defined exception handling technique [10, 17], user specifies the particular treatment to workflow of a task on failure.
4. *Rescue workflow*—The rescue workflow technique [10] 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 grid computing, resource management and job scheduling are classified into two broad categories: (a) system-centric and (b) user-centric. The system-centric approach [18] aims to maximize the overall system utilization by considering the parameters such as throughput, turnaround time, transmission delay, waiting time, etc. On the other hand, user-centric approach focuses on meeting the QoS requirements such as duration and cost of computation. As most of the grid resource management and scheduling systems, such as Condor, AppLeS PST, PUNCH, and Netsolve, working on system-centric approach [18, 19], we also identify system-centric parameters for measuring performance in our proposed strategy.

A *gridlet* is a package that contains all the information related to the job and its execution management details such as job length expressed in MI (Millions Instruc-

tion), the size of input and output files, and the job owner ID. Individual users model their application by creating gridlets for processing them on grid resources.

The rest of the paper is organized as follows. In Sect. 2, we discuss the problem statement along with the proposed solution to the problem. Section 3 contains the methodology that we have adopted. Section 4 comprises the experimental results along with their discussion, while in Sect. 5 we draw conclusion of our study.

2 Fault tolerance in grid computing

2.1 Problem definition

Alternate task technique is one of the important FTTs at workflow level. Alternate task technique exchanges a task with other implementations of the same task but with different execution characteristics on failure of the first one. For example, suppose that a fault occurs due to memory overflow. Then the alternate task technique will resubmit the second implementation of the same task which would require less memory load as compared to the first one. Here while initiating the second implementation, alternate task technique assumes that this task will not fail again and it has no mechanism to handle a fault the second time. But the resubmitted task can fail again due to some other reason such as thread conflicts or out-of-disk space on that resource. In this case, alternate task technique does not provide any mechanism to handle a failure that occurs the second time due to some other reason or even due to the same reason because it has only one alternative implementation of that task which it has already applied.

On the other hand, checkpoint technique being the task level FTT does not consider the reason of a failure. Suppose that a resource available to us does not have the capability to handle a specific type of a fault such as out-of-memory or disk space to execute the task. In this case, simple checkpoint technique will never be able to finish the execution of the task.

In this paper we consider the problems of workflow level alternate task technique and task level checkpoint technique. We provide a novel and efficient FTT combining the two techniques. We call it *alternate task with checkpoint.* This technique overcomes the limitations of alternate task and checkpoint FTTs and inherits the best features of both, as is shown in Fig. 1. It shows a significant improvement over other techniques under different conditions considered in our experiments, such as different workloads with different percentages of faults injected in a system. In our study, we consider different performance parameters such as throughput, turnaround time, waiting time and network delay.

2.2 Alternate task with retry

The simplest solution to this problem is to hybrid a task level FTT and workflow level FTT. This is to avoid failures at both task level and workflow level separately. *Alternate task with retry* is a hybrid of "alternate task" FTT at workflow level and "retry" FTT at task level. After the failure of an alternate task, alternate task with

FTT Name	Saves Intermediate Results	Consider Multiple Implementations	Consider Multiple Resubmissions	Consider Alternate Resource
Alternate Task	**No**	**Yes**	**No**	**No**
Checkpoint	**Yes**	**No**	**Yes**	**Yes**

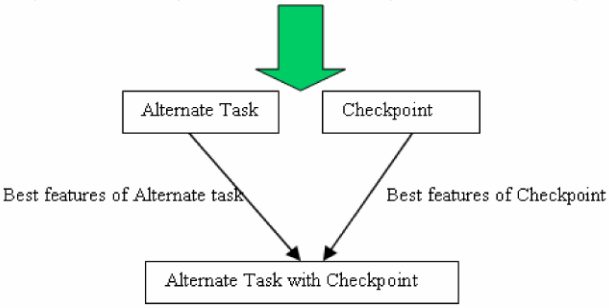

Fig. 1 (Color online) Features of alternate task with checkpoint

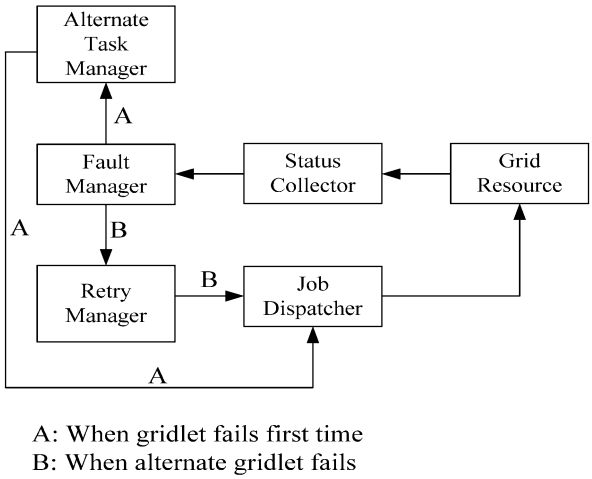

Fig. 2 Framework of alternate task with retry

retry FTT simply retries the failed alternate task on the same resource up to a certain threshold value which is in our case three. In this way we can overcome failures of a system to a certain extent. However, it is not an efficient solution to this problem, as our results show in the following sections. Figure 2 shows the framework of this technique and different steps involved in it. The process is given in Algorithm 1.

Algorithm 1 Alternate_Task_with_Retry ( )

1. STATUS COLLECTOR: Collects the status of failed gridlet from the grid resource and forwards the gridlet_ID to the fault manager

2. FAULT MANAGER: Receives the failed gridlet_ID. If a gridlet fails the first time, it interacts with alternate task manager. If it is an alternate gridlet, it interacts with the retry manager.
   a. ALTERNATE TASK MANAGER: If the gridlet fails the first time, then it is forwarded to the alternate task manager. The alternate task manager selects an appropriate alternative gridlet and sends it to the job dispatcher.
   b. RETRY MANAGER: If it is an alternate gridlet, then it is forwarded to the retry manager. The retry manager verifies the number of submissions of the alternate gridlet. If it is within the threshold value, it forwards the alternate gridlet to the grid dispatcher.
3. JOB DISPATCHER: Receives the gridlet and resubmits it to the same grid resource.

2.3 Alternate task with checkpoint

In this FTT we choose task level checkpoint FTT with workflow level alternate task FTT in order to minimize the failures. When a task fails the first time, alternate task manager finds an appropriate alternate task and forwards it to the job dispatcher. The job dispatcher submits the alternate task to the same resource. When the alternate task fails the first time, the checkpoint manager applies checkpoints to the task and forwards it to the job dispatcher. The job dispatcher submits it to the same grid resource. When the alternate task fails again, the checkpoint manager retrieves the intermediate results of the last saved checkpoint from the checkpoint information server (CIS) and forwards the incomplete gridlet with intermediate results to the job dispatcher. The job dispatcher in turn submits the incomplete gridlet and intermediate results to another suitable grid resource. The details are in Algorithm 2.

Algorithm 2 Alternate_Task_with_Checkpoint ( )

1. STATUS COLLECTOR: Collects the status of failed gridlet from the grid resource and forwards the gridlet_ID to the fault manager
2. FAULT MANAGER: Receives the failed gridlet_ID. If the failed gridlet is not an alternate gridlet, it interacts with alternate task manager. Otherwise, it interacts with the checkpoint manager.
   a. ALTERNATE TASK MANAGER: When a gridlet fails the first time, it is forwarded to alternate task manager. The alternate task manager selects an appropriate alternative gridlet and sends it to the job dispatcher. The job dispatcher in turn submits to the same grid resource.
   b. CHECKPOINT MANAGER: When an alternate gridlet fails the first time, it is forwarded to the checkpoint manager. When the alternative gridlet is received the first time, the checkpoint manager assigns appropriate checkpoints to the alternative gridlet and forwards it to the job dispatcher. The job dispatcher in turn submits to the same grid resource.
   When the alternate gridlet fails again, the checkpoint manager retrieves the intermediate results of the last saved checkpoint from CIS and forwards the incomplete gridlet with intermediate results to the job dispatcher. The job dispatcher in turn submits the incomplete gridlet and intermediate results to another suitable grid resource.

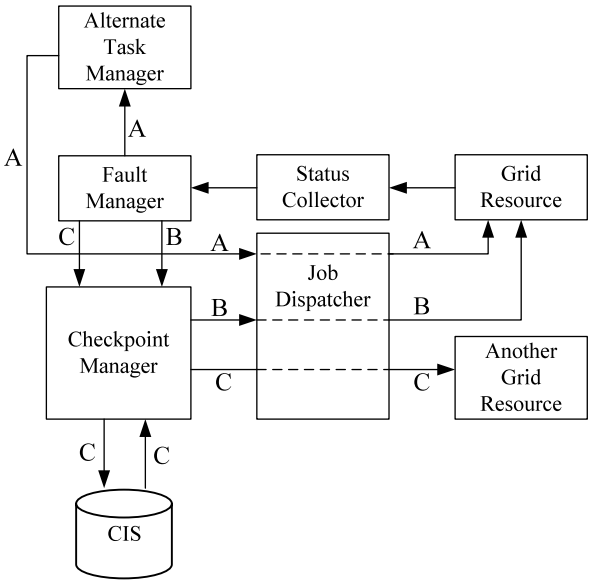

Fig. 3 Framework of alternate task with checkpoint

A: When a gridlet fails first time
B: When an alternate gridlet fails first time
C: When the alternate gridlet fails again

Figure 3 shows the framework of alternate task with checkpoint and different steps involved in it. Alternate task with checkpoint FTT overcomes the limitations of alternate task FTT and checkpoint FTT. Moreover, it inherits the good features of both. See Fig. 1. Our proposed technique has the following distinct features:

- It saves intermediate results unlike the simple alternate task technique.
- It considers different task unlike the simple checkpoint technique.
- It considers multiple resubmissions unlike the simple alternate task technique.
- It also considers an alternate resource unlike the simple alternate task technique.

3 Experimental methodology

We carry out our study by means of GridSim simulator [19]. GridSim supports modeling and simulation of heterogeneous grid resources, users and application models. It provides infrastructure for creation of application tasks, mapping of tasks to resources, and their management.

In our study we model workflow level FTTs and measure the performance of each FTTs on parameters defined in Sect. 3.1. The experiment setup is the same for all the FTTs. In this study we model and create grid resources and applications that model jobs as gridlets in GridSim environment.

3.1 Resource and application modeling

In our experiments we model different space-shared resources and use the configurations, characteristics, and capabilities of these resources from WWG testbed [20]. In our experimental setup, the number of jobs of an application varies from 200 to 1000 and the communication link is 56 Mbps (mega-bits per second). The size of input file is 300 MB and job length is 10000 MI for all jobs. These specifications remain the same for all FTTs.

3.2 Performance metrics

Here is the list of performance metrics in our experiments:

1. **Average throughput**—*Throughput* is defined in general as a number of gridlets completed per time unit.

$$\text{Throughput} = n/T$$

   where n is number of gridlets submitted and T is total amount of time to complete the n gridlets successfully.
2. **Average turnaround time**—The interval from the time of submission of a gridlet to the time of completion is the *turnaround time*. The average turnaround time is the average over all gridlets submitted (see Fig. 4).
3. **Average waiting time**—The *waiting time* is the amount of time between submitting a gridlet to a grid resource and starting the execution of the gridlet by the grid resource (see Fig. 4).

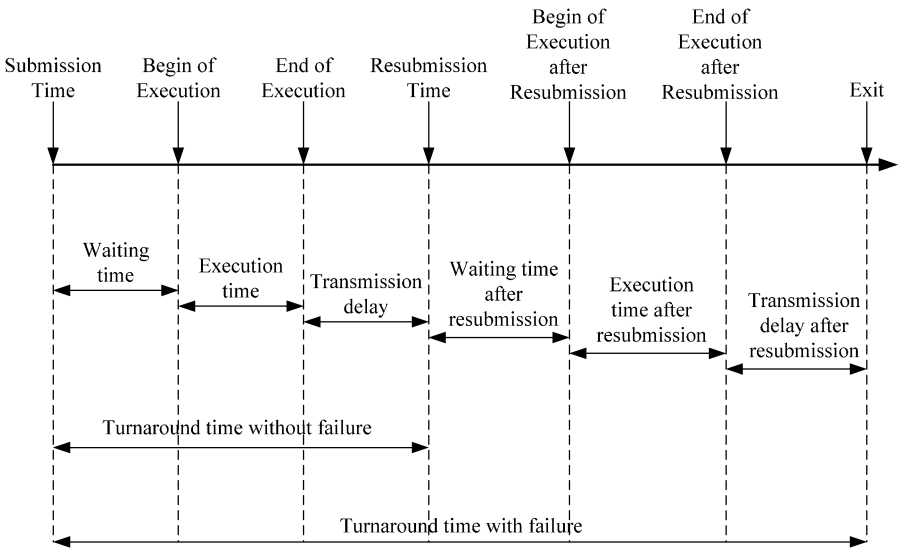

Fig. 4 Definition of timing parameters

4. **Transmission delay**—*Transmission delay* is the amount of time required to push all of the output files (in bits) back to the grid user. Transmission delay [21] is a function of the size of output file. It is given by the formula:

$$D_T = N/R$$

where, D_T is the transmission delay, N is the size of output file (bits), and R is the rate of transmission (bits per second).

4 Experimental results and discussion

In this section we present the results of our simulation. Average throughput, average turnaround time, average waiting time, and average transmission delay of a system are recorded from different experiments under different conditions using both alternate task with retry and alternate task with checkpoint FTTs.

4.1 Average throughput

Throughput is an important parameter for determining the performance of different FTTs. It becomes more important when a user has to wait for the completion of all jobs before proceeding further. Figures 5, 6, 7, 8, 9 depict average throughput of alternate task with retry and alternate task with checkpoint FTTs.

Figures 5–9 depict throughput of both FTTs with different percentage of faults injected in a system, and different number of gridlets. With different percentage of faults injected and different number of gridlets submitted, the overall difference between these two FTTs for average throughput is a little more than 32%.

In general, the average throughput of both techniques decreases with increase in the percentage of faults injected and with the increase in the number of gridlets submitted to the system. Figures 5–9 show that the throughput for alternate task with

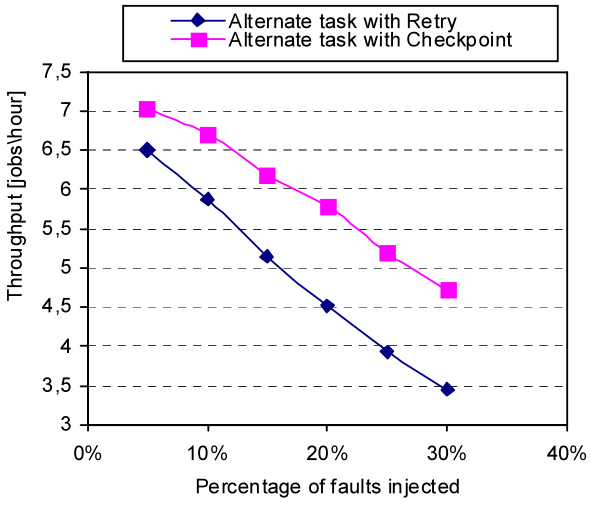

Fig. 5 (Color online) Total number of gridlets/job submitted = 200

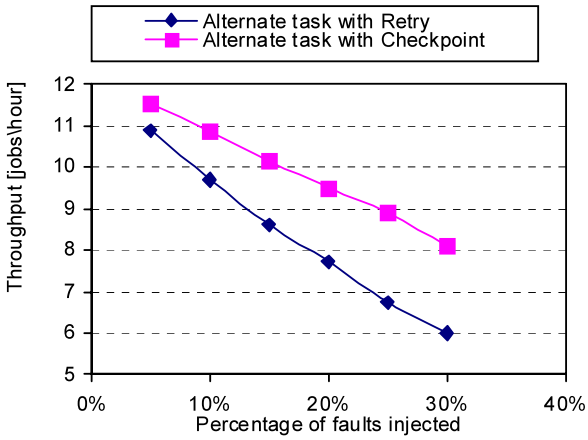

Fig. 6 (Color online) Total number of gridlets/job submitted = 400

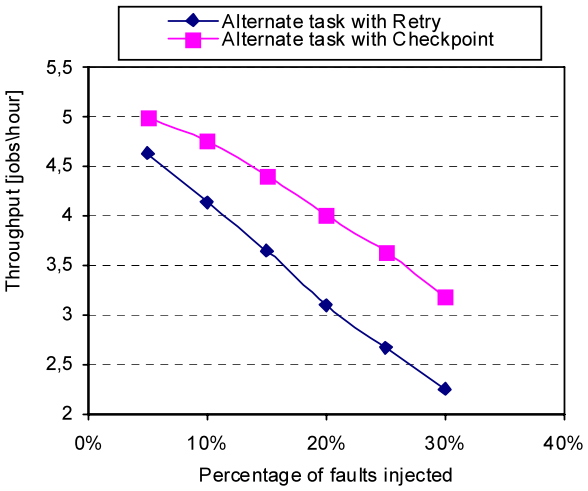

Fig. 7 (Color online) Total number of gridlets/job submitted = 600

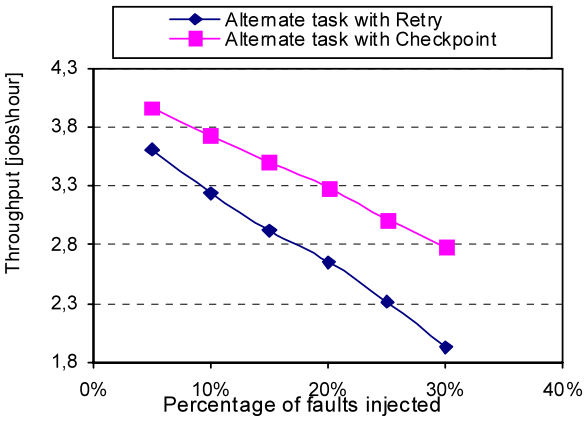

Fig. 8 (Color online) Total number of gridlets/job submitted = 800

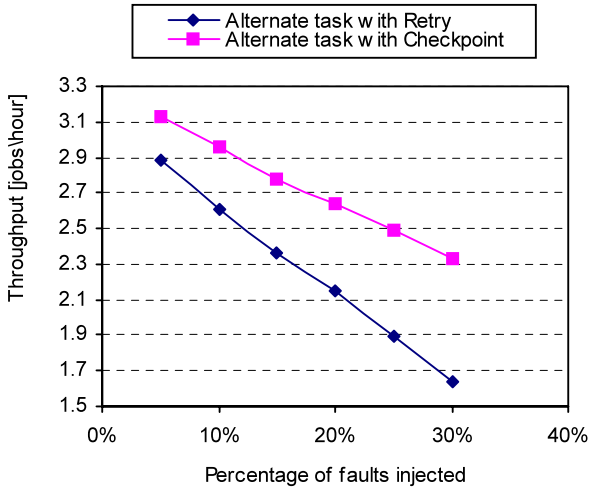

Fig. 9 (Color online) Total number of gridlets/job submitted = 1000

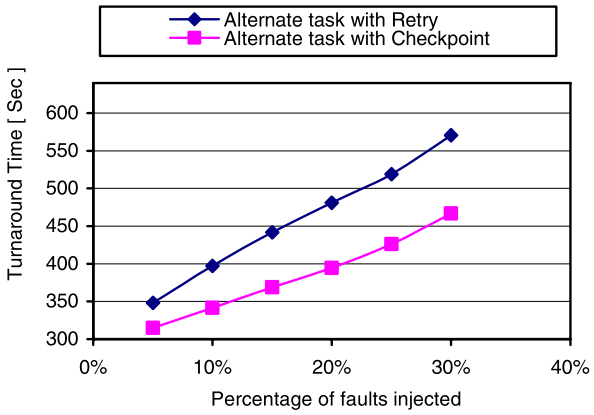

Fig. 10 (Color online) Total number of gridlets/job submitted = 200

checkpoint is larger in all conditions than alternate task with retry. This trend becomes more visible when we increase the percentage of faults injected in a system and with increase in the number of gridlets submitted to a system.

The increase in throughput of alternate task with checkpoint is due to a failure of resubmitted gridlet of an alternate task. In case of alternate task with retry, the resubmitted gridlet has to start its execution right from the beginning on every resubmission which would ultimately increase the overall execution time, transmission delay, and waiting time for that particular gridlet. Hence the overall performance of a system decreases. In the case of alternate task with checkpoint, the resubmitted task starts its execution from last saved checkpoint. Storing the intermediate results in case of checkpoint saves the extra execution time and decreases the waiting time as well as the transmission delay. These entire factors combine together to increase the overall throughput of a system as depicted in Figs. 5–9.

Table 1 shows the detailed relative throughput in percentage of both FTTs for different percentage of faults injected in a system with different workloads. From

Table 1 Relative performance of FTTs with respect to average throughput (B—relatively best, W—relatively worst performance. Atc—Alternate task with checkpoint, Atr—Alternate task with retry)

No. of gridlets	Percentage of faults injected in a system					
	5%	10%	15%	20%	25%	30%
200	B: Atc W: Atr (5.42%)	B: Atc W: Atr (10.7%)	B: Atc W: Atr (15.2%)	B: Atc W: Atr (18.9%)	B: Atc W: Atr (24.3%)	B: Atc W: Atr (26.1%)
400	B: Atc W: Atr (7.471%)	B: Atc W: Atr (12.4%)	B: Atc W: Atr (16.9%)	B: Atc W: Atr (22%)	B: Atc W: Atr (24.3%)	B: Atc W: Atr (27.06%)
600	B: Atc W: Atr (7.42%)	B: Atc W: Atr (13%)	B: Atc W: Atr (17.3%)	B: Atc W: Atr (22.6%)	B: Atc W: Atr (26.6%)	B: Atc W: Atr (29.2%)
800	B: Atc W: Atr (8.935%)	B: Atc W: Atr (13.2%)	B: Atc W: Atr (16.6%)	B: Atc W: Atr (19.2%)	B: Atc W: Atr (23.1%)	B: Atc W: Atr (30.48%)
1000	B: Atc W: Atr (7.826%)	B: Atc W: Atr (11.7%)	B: Atc W: Atr (15.1%)	B: Atc W: Atr (18.7%)	B: Atc W: Atr (24.2%)	B: Atc W: Atr (32.34%)

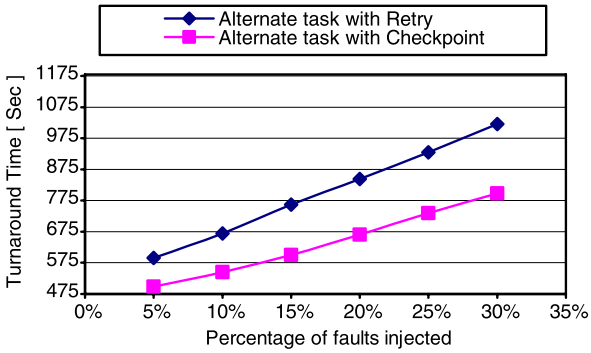

Fig. 11 (Color online) Total number of gridlets/job submitted = 400

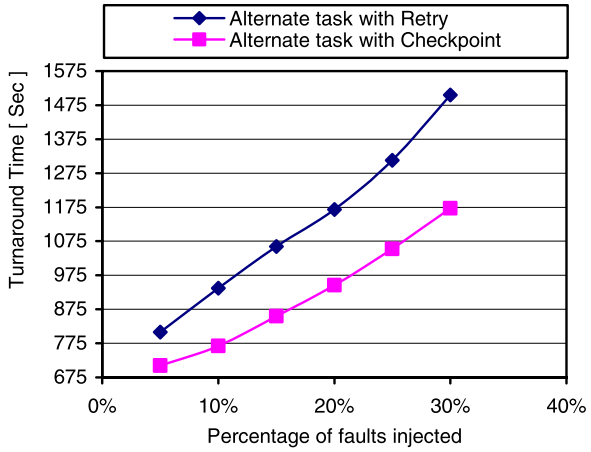

Fig. 12 (Color online) Total number of gridlets/job submitted = 600

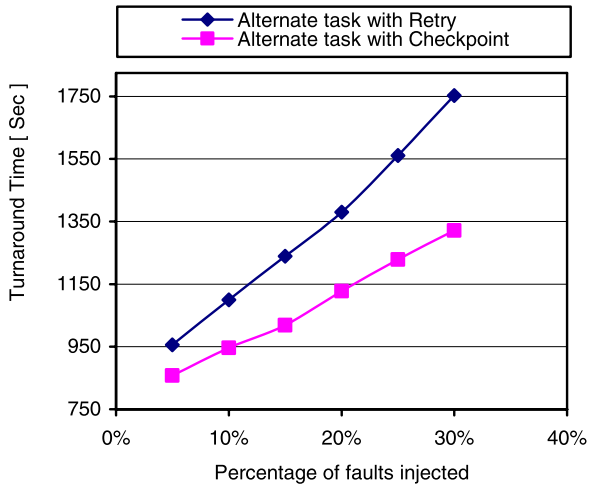

Fig. 13 (Color online) Total number of gridlets/job submitted = 800

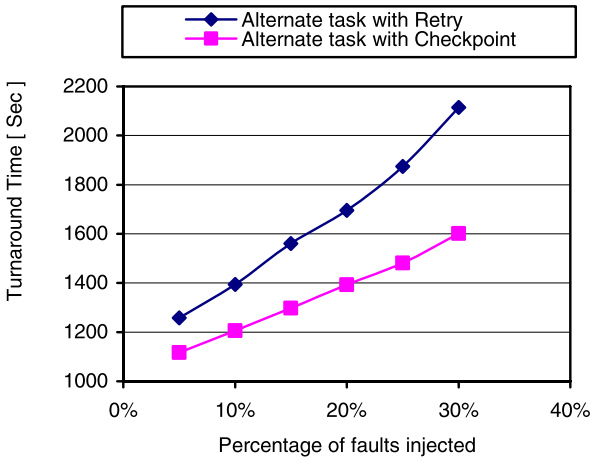

Fig. 14 (Color online) Total number of gridlets/job submitted = 1000

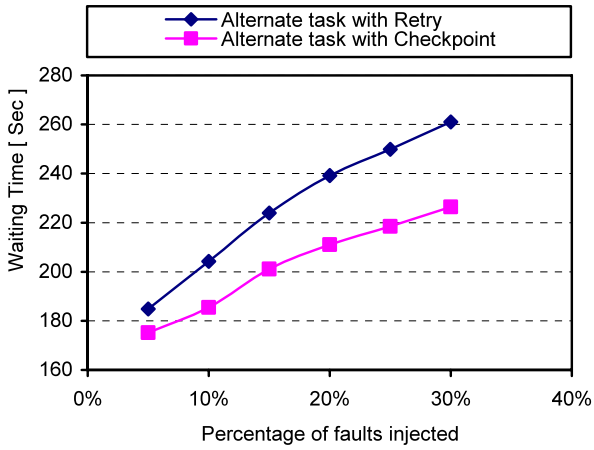

Fig. 15 (Color online) Total number of gridlets/job submitted = 200

this table we can easily detect the improvement in percentage at different sorts of conditions.

4.2 Average turnaround time

Figures 10, 11, 12, 13, 14 depict turnaround time of both alternate task with checkpoint and alternate task with retry FTTs with different percentage of faults injected in a system and different number of gridlets submitted. The overall difference between both FTTs is almost 34% for average turnaround time.

In case of alternate task with retry, the resubmitted gridlet has to start its execution right from the beginning on every resubmission, which ultimately increases the transmission delay for that particular gridlet. In case of alternate task with checkpoint, the resubmitted task has to start its execution from the last saved checkpoint. Storing the intermediate results saves the extra execution time and decreases the transmission delay. As in Figs. 10–14, the turnaround time is reciprocal to throughput.

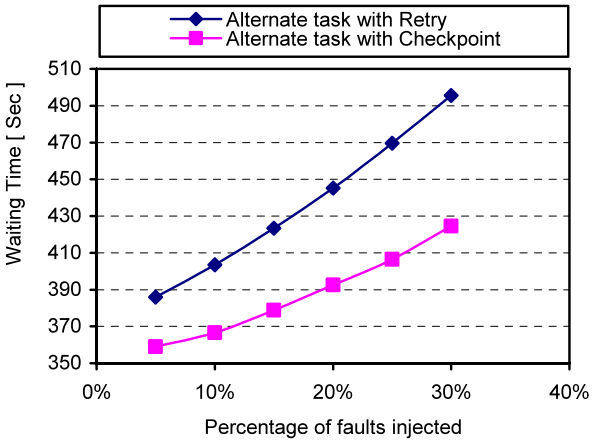

Fig. 16 (Color online) Total number of gridlets/job submitted = 400

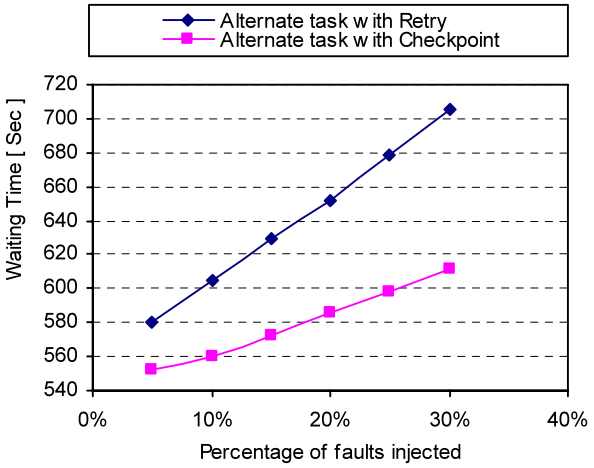

Fig. 17 (Color online) Total number of gridlets/job submitted = 600

Table 2 shows the detailed relative turnaround time in percentage of both FTTs for different percentages of faults injected in a system with different workloads. From the table, the alternate task with checkpoint FTT is superior to alternate task with retry.

4.3 Average waiting time

Figures 15, 16, 17, 18, 19 illustrate 'Waiting Time' of both alternate task with checkpoint and alternate task with retry FTTs with different percentage of faults injected in a system and different number of gridlets submitted. With different percentage of faults injected and number of gridlets submitted, the overall difference between both FTTs is almost 18% for average waiting time. Table 3 shows the detailed relative waiting time in percentage of both FTTs for different percentage of faults injected in a system with different workloads. From this table we can easily conclude that the alternate task with checkpoint is a better choice.

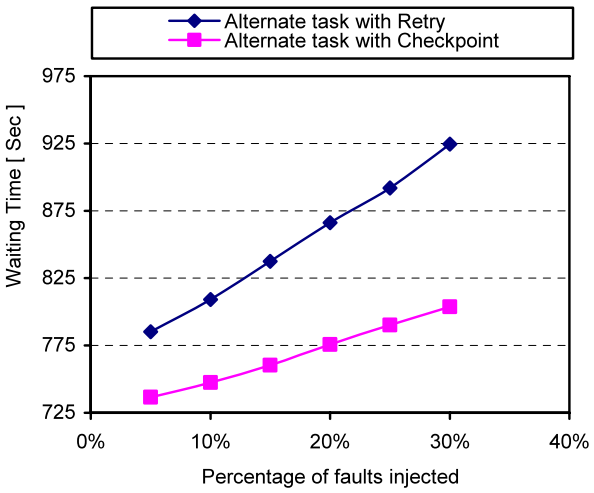

Fig. 18 (Color online) Total number of gridlets/job submitted = 800

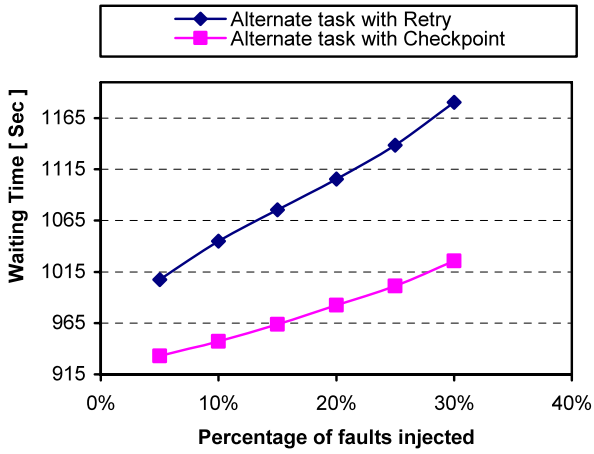

Fig. 19 (Color online) Total number of gridlets/job submitted = 1000

4.4 Average transmission delay

Figures 20, 21, 22, 23, 24 depict an average transmission delay of both FTTs. With different percentage of faults injected and number of gridlets submitted, the overall difference between both FTTs is more than 14% for average transmission delay. Table 4 shows the detailed relative transmission delays in percentage of both FTTs for different percentages of faults injected in a system with different workloads.

5 Conclusion

The relative performance of both FTTs under different conditions that we have considered in our investigation is given in the form of a summary in Tables 1, 2, 3 and 4. In each table the performance is examined with respect to parameter specified.

Table 2 Relative performance of FTTs with respect to Average Turnaround time (B—relatively best, W—relatively worst performance. Atc—Alternate Task with checkpoint, Atr—Alternate task with retry)

No. of gridlets	Percentage of faults injected in a system					
	5%	10%	15%	20%	25%	30%
200	B: Atc W: Atr (−10.5%)	B: Atc W: Atr (−16%)	B: Atc W: Atr (−20%)	B: Atc W: Atr (−22%)	B: Atc W: Atr (−22%)	B: Atc W: Atr (−22.3%)
400	B: Atc W: Atr (−18.4%)	B: Atc W: Atr (−23%)	B: Atc W: Atr (−27%)	B: Atc W: Atr (−27%)	B: Atc W: Atr (−27%)	B: Atc W: Atr (−27.8%)
600	B: Atc W: Atr (−13.8%)	B: Atc W: Atr (−22%)	B: Atc W: Atr (−24%)	B: Atc W: Atr (−24%)	B: Atc W: Atr (−25%)	B: Atc W: Atr (−28.4%)
800	B: Atc W: Atr (−11.4%)	B: Atc W: Atr (−16%)	B: Atc W: Atr (−22%)	B: Atc W: Atr (−22%)	B: Atc W: Atr (−27%)	B: Atc W: Atr (−32.7%)
1000	B: Atc W: Atr (−12.6%)	B: Atc W: Atr (−16%)	B: Atc W: Atr (−20%)	B: Atc W: Atr (−22%)	B: Atc W: Atr (−26%)	B: Atc W: Atr (−33.7%)

Table 3 Relative performance of FTTs with respect to average waiting Time (B—relatively best, W—relatively worst performance. Atc—Alternate task with checkpoint, Atr—Alternate task with retry)

No. of gridlets	Percentage of faults injected in a system					
	5%	10%	15%	20%	25%	30%
200	B: Atc W: Atr (−5.56%)	B: Atc W: Atr (−10%)	B: Atc W: Atr (−11%)	B: Atc W: Atr (−13%)	B: Atc W: Atr (−14%)	B: Atc W: Atr (−15.3%)
400	B: Atc W: Atr (−7.52%)	B: Atc W: Atr (−10%)	B: Atc W: Atr (−12%)	B: Atc W: Atr (−13%)	B: Atc W: Atr (−16%)	B: Atc W: Atr (−16.8%)
600	B: Atc W: Atr (−5%)	B: Atc W: Atr (−8%)	B: Atc W: Atr (−9.9%)	B: Atc W: Atr (−11%)	B: Atc W: Atr (−13%)	B: Atc W: Atr (−16.9%)
800	B: Atc W: Atr (−6.61%)	B: Atc W: Atr (−8.3%)	B: Atc W: Atr (−10%)	B: Atc W: Atr (−12%)	B: Atc W: Atr (−13%)	B: Atc W: Atr (−17.3%)
1000	B: Atc W: Atr (−9.09%)	B: Atc W: Atr (−11%)	B: Atc W: Atr (−13%)	B: Atc W: Atr (−14%)	B: Atc W: Atr (−15%)	B: Atc W: Atr (−17.6%)

Table 4 Relative performance of FTTs with respect to average transmission delay (B—relatively best, W—relatively worst performance. Atc—Alternate Task with checkpoint, Atr—Alternate task with retry)

No. of gridlet	Percentage of faults injected in a system					
	5%	10%	15%	20%	25%	30%
200	B: Atc W: Atr (−3.17%)	B: Atc W: Atr (−3.56%)	B: Atc W: Atr (−6.05%)	B: Atc W: Atr (−7.7%)	B: Atc W: Atr (−9.4%)	B: Atc W: Atr (−10.1%)
400	B: Atc W: Atr (−5.1%)	B: Atc W: Atr (−6.65%)	B: Atc W: Atr (−6.6%)	B: Atc W: Atr (−7.9%)	B: Atc W: Atr (−9.7%)	B: Atc W: Atr (−11.3%)
600	B: Atc W: Atr (−5.29%)	B: Atc W: Atr (−6.57%)	B: Atc W: Atr (−8.53%)	B: Atc W: Atr (−9.5%)	B: Atc W: Atr (−11%)	B: Atc W: Atr (−11.9%)
800	B: Atc W: Atr (−4.52%)	B: Atc W: Atr (−5.96%)	B: Atc W: Atr (−7.47%)	B: Atc W: Atr (−9%)	B: Atc W: Atr (−10%)	B: Atc W: Atr (−13.1%)
1000	B: Atc W: Atr (−4.58%)	B: Atc W: Atr (−6.13%)	B: Atc W: Atr (−9.55%)	B: Atc W: Atr (−10%)	B: Atc W: Atr (−12%)	B: Atc W: Atr (−14.5%)

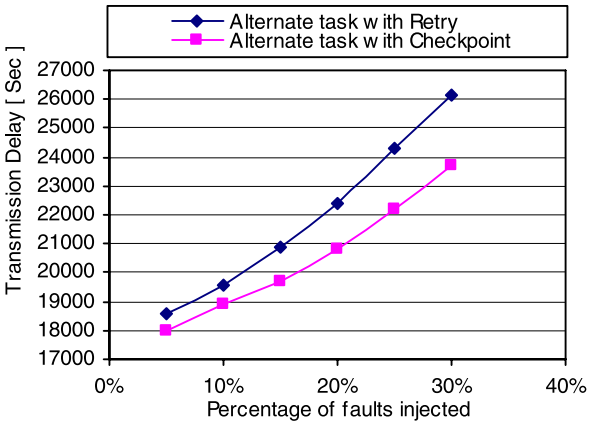

Fig. 20 (Color online) Total number of gridlets/job submitted = 200

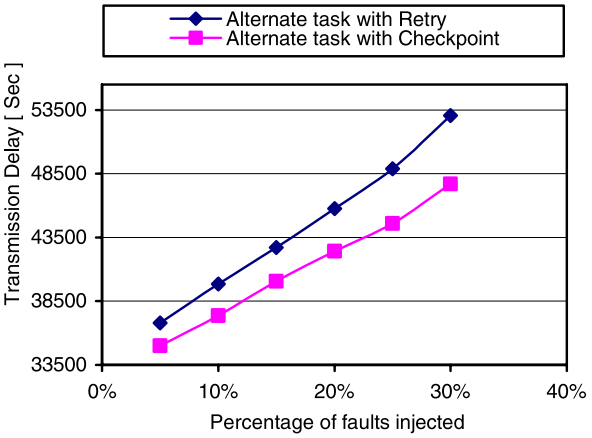

Fig. 21 (Color online) Total number of gridlets/job submitted = 400

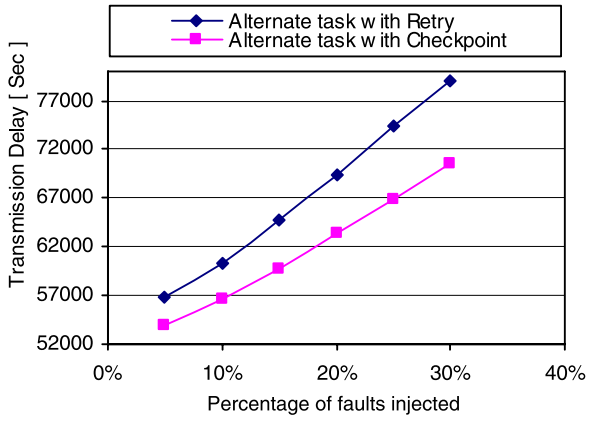

Fig. 22 (Color online) Total number of gridlets/job submitted = 600

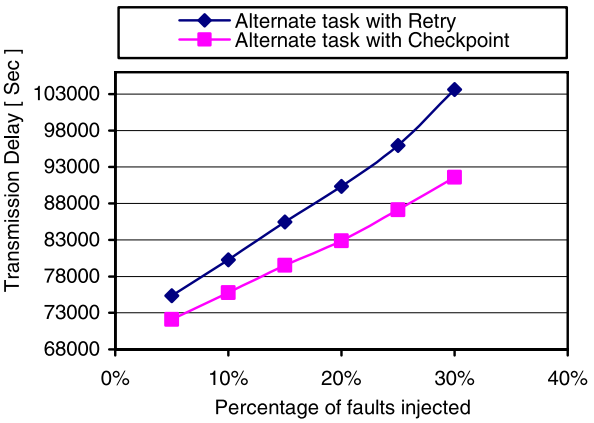

Fig. 23 (Color online) Total number of gridlets/job submitted = 800

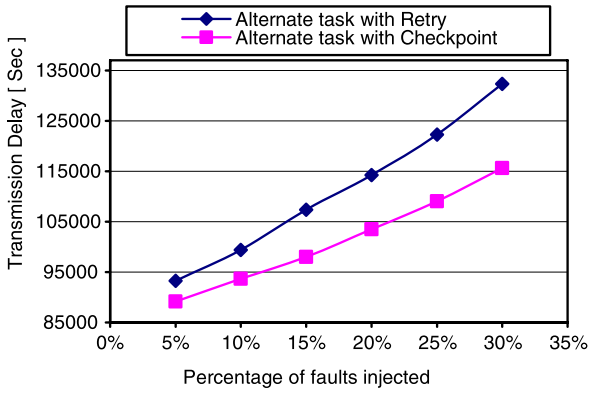

Fig. 24 (Color online) Total number of gridlets/job submitted = 1000

In this paper we proposed a novel hybrid FTTs and configured its framework in grid computing. We evaluated the performance of our techniques under different conditions using different parameters such as throughput, turnaround time, waiting time, and transmission delay. We found out that alternate task with checkpoint yields better results in all conditions than alternate task with retry.

References

1. Jankowski G, Januszewski R, Mikolajczak R, Kovacs J (2008) Improving the fault tolerance level within the GRID computing environment-integration with the low-level checkpointing packages. CoreGRID Technical Report Number TR-0158, June 16
2. Chandy KM, Lamport L (1985) Distributed snapshots: determining global states of distributed systems. ACM Trans Comput Syst (February), 63–75
3. Mattern F (1993) Efficient algorithms for distributed snapshots and global virtual time approximation. J Parallel Distrib Comput, 423–434
4. Abawajy JH (2004) Fault-tolerant scheduling policy for grid computing systems. In: 18th International parallel and distributed processing symposium (IPDPS'04)—workshop 13, 2004, vol 14, p 238b
5. Lee H, Chung K, Chin S, Lee J, Lee D, Park S, Yu H (2005) A resource management and fault tolerance services in grid computing. J Parallel Distrib Comput 65(11):1305–1317

6. Hwang S, Kesselman C (2004) A flexible framework for fault tolerance in the grid. J Grid Comput 1(3):251–272. doi:10.1023/B:GRID.0000035187.54694.75
7. 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, April 26–30, 2004. IEEE Computer Society Press, Los Alamitos, pp 238–244
8. Yu J, Buyya R (2005) A taxonomy of workflow management systems for grid computing. J Grid Comput 3(3–4):171–200. doi:10.1007/s10723-005-9010-8
9. Gartner FC (1999) Fundamentals of fault-tolerant distributed computing in asynchronous environments. J ACM Comput Surv 31(1):1–26
10. Anglano C, Canonico M (2005) Fault-tolerant scheduling for bag-of-tasks grid applications. In: Advances in grid computing—EGC 2005. Lecture notes in computer science, vol 3470/2005. Springer, Berlin/Heidelberg. ISSN: 0302-9743 Print. doi:10.1007/b137919, ISBN: 978-3-540-26918-2, pp 630–639
11. 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 Press, Los Alamitos, ISSN: 0-7695-3064-8
12. Gioiosa R, Sancho JC, Jiang S, Petrini F, Davis K (2005) Incremental check-pointing at kernel level: a foundation for fault tolerance for parallel computers. In: Proceedings of the 2005 ACM/IEEE SC|05 conference (SC'05)
13. Hwang S, Kesselman C (2003) Grid workflow: a flexible failure handling framework for the grid. In: 12th IEEE international symposium on high performance distributed computing (HPDC'03), Seattle, Washington, USA, June 22–24, 2003. IEEE Computer Society Press, Los Alamitos
14. Buyya R (2002) Economic-based distributed resource management and scheduling for grid computing, Ph.D. Thesis, Monash University, Melbourne, Australia, April 12
15. Fahringer T et al (2005) Truong. ASKALON: a tool set for cluster and Grid computing. J Concurr Comput Pract Exp 17(2–4):143–169
16. von Laszewski G (2006) Workflow Concepts of the Java CoG Kit. J Grid Comput 3(3–4):239–258
17. Ludascher B et al (2006) Scientific workflow management and the KEPLER system. J Concurr Comput Pract Exp 18(10):1039–1065
18. Yu J, Buyya R (2004) A novel architecture for realizing grid workflow using tuple spaces. In: 5th IEEE/ACM international workshop on grid computing (GRID 2004), Pittsburgh, USA, 2004. IEEE Computer Society Press, Los Alamitos, ISBN: 0-7695-2256-4
19. Buyya R, Murshed M (2002) GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing. J Concurr Comput Pract Exp 13(13–15)
20. Testbed WWG (2008) http://gridbus.cs.mu.oz.au/sc2003/list.html [August 2008]
21. Nazir B, Qureshi K, Manuel P (2008) Adaptive checkpointing strategy to tolerate faults in economy based grid, J Supercomput. doi:10.1007/s11227-008-0245-6

Kalim Qureshi is a faculty member of Information Science Department, Kuwait University and he is adjunct Professor in Computer Science Department, COMSATS Institute of Information Technology, Abbottabad, Pakistan. He is an approved supervisor for the M.Sc. and Ph.D. theses by the High Education Commission, Islamabad, Pakistan. His research interests include network parallel distributed computing, thread programming, concurrent algorithms design, task scheduling, and performance measurement. Dr. Qureshi is a member of IEE Japan and IEEE Computer Society. His e-mail addresses are kalim@ciit.net.pk and kalim_qureshi@hotmail.com.

Fiaz Gul Khan is a Ph.D. student in Wireless Systems and Related Technologies at University of Politecnico di Torino, Italy. His research interests include distributed computing, wireless networks, network architecture, communication protocols, and simulation tools. He received his Master's degree from COMSATS Institute of Information technology Abbottabad, Pakistan. Contact him at fiazgul.khan@studenti.polito.it.

Paul Manuel is Associate Professor in Information Science in Kuwait University. He graduated with a M.Sc. in Computer Science from the University of Saskatchewan, Canada, and has a Ph.D. in Computer Science from the University of Newcastle, Australia. His second Ph.D. is in Mathematics from the Indian Institute of Technology, Chennai, India. His research area includes grid computing, software engineering, and graph theory. He has published more than 50 research papers in internationally reputed journals.

Babar Nazir currently is a Ph.D. student at University of Teknologi PETRONAS, Malaysia. He is a faculty member of COMSATS Institute of Information Technology, Abbottabad Campus, Pakistan. He received his M.Sc. in Computer Science from COMSATS Institute of Information Technology, Abbottabad Campus, Pakistan, in 2007. He published three international conference papers and two journal papers in reputed journals. His research interests include resource management and job scheduling in grid computing, parallel and distributed computing, wireless sensor networks and mobile ad hoc networks. His e-mail addresses are babarnazir@ciit.net.pk and babarnazir@gmail.com.