IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 3, 2017, accepted March 17, 2017, date of publication April 3, 2017, date of current version June 7, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2690458

Dynamic and Adaptive Fault Tolerant Scheduling With QoS Consideration in Computational Grid

SAJJAD HAIDER[1,2], AND BABAR NAZIR[3]

[1]Department of Computing, Shaheed Zulfiqar Ali Bhutto Institute of Science and Technology, Islamabad 44000, Pakistan
[2]Department of Computer Science, National University of Modern Languages, Islamabad 44000, Pakistan
[3]Department of Computer Science, COMSATS Institute of Information Technology, Abbottabad 22010, Pakistan

Corresponding author: Sajjad Haider (sajjadhyder@hotmail.com)

ABSTRACT Provisioning fault-tolerant scheduling in computational grid is a challenging task. Most of the existing fault tolerant scheduling schemes are either geared toward proactive or reactive. Proactive schemes emphasize on the reasons responsible for generating faults, whereas reactive mechanisms come into effect after failure detection. Unlike most existing mechanisms, we present a novel, dynamic, adaptive, and hybrid fault-tolerant scheduling scheme based on proactive and reactive approaches. In the proactive approach, the resource filtration algorithm picks resources based on resource location, availability, and reliability. Unlike most existing schemes, which rely on remotely connected resources, the proposed algorithm prefers to employ locally available resources as they might have less failure tendency. To cope with the frequent turnover problem, the proposed scheme calculates resource availability time based on various newly identified parameters (e.g., mean time between availability) and picks highly available nodes for task execution. Resource reliability is an indispensable consideration in the proposed scheme and is calculated based on parameters such as jobs success or failure ratio and the types of failures encountered. We employ an optimal resource identification algorithm to determine and select optimal resources for job execution. The performance of the proposed scheme is validated through the GridSim toolkit. Compared with contemporary approaches, experimental results demonstrate the effectiveness and efficiency of the proposed scheme in terms of various performance metrics, such as wall clock time, throughput, waiting and turnaround time, number of checkpoints, and energy consumption.

INDEX TERMS Computational grid, fault tolerant scheduling, availability, reliability, genetic algorithm.

I. INTRODUCTION

Computational grid is a network where locally and remotely connected machines offer their idle CPU cycles to be used by grid users for execution of their compute intensive jobs. Grid is a complex communication system where heterogeneous resources are available for use under a decentralized management. Such type of arrangement creates problems for availability and reliability of resources and failures become the norm [1] rather than the exception [2]. Economic benefit of grid is that resource owners get financial gains when their machine or resource is utilized by other grid users. So, if a resource in grid is reliable and is consumed most of the time, then it's a good source of income for the owner of that resource. Cloud on the other hand is maintained by cloud service providers and more the users utilize cloud services, more they pay for it.

One of the important components of a grid is the resource manager, as it is responsible for managing resources. Jobs executing under grid require resources whose efficient, reliable and careful selection and scheduling with respect to fault tolerance may improve overall performance. It is the responsibility of a scheduler to map task on suitable and appropriate grid resources [3]. Resource management duties are not just limited to provisioning of resources but to ascertain an accepted level of Quality of Service (QoS). Due to this reason it is important to constantly monitor resource allocation so that QoS can be maintained [4], [5]. Variation in bandwidth and network delays can further be a hindrance in enhancement of computing performance of grid [6]. With the increase in the number of resources from various sites, the probability of resource failure increases. Fault tolerance becomes a mandatory task to be performed in failure prone distributed systems like clusters, grids and clouds [7]. Dependability of grid can be improved through identification of reliable and available resources under a fault tolerant scheduling design [8].

Fault Tolerance (FT) is the capability developed in the system to continue working despite the presence of failures. Proactive and reactive fault tolerant schemes are widely used by the researchers in order to handle the issue. Reactive or post-active schemes are applied after the detection of failure. An effective failure detection technique in this case may improve failure handling mechanism. Due to the complexity of grid there are number of reasons that advocate failures of different types. A real challenge is that all types of failures cannot be incorporated in fault tolerance designs due to complexity, performance and the fact that there could be other possible reasons of failures not identified or explored yet. Proactive fault tolerance is considered effective as it focuses on the techniques that minimize the chances of failures through identification of less failure prone resources. A combination of proactive and reactive fault tolerance technique with fault detection mechanism can improve overall productivity of High Performance Computing (HPC) systems like grid and cloud.

In this paper we propose a dynamic and adaptive fault tolerant (DFT) scheduling for computational grids. The model consists of two main components, i) Proactive and ii) Reactive FT orchestrators. Proactive FT orchestrator consists of two modules, i) resource filtration module and ii) optimal resource selection module. The former is an ongoing process in which grid resources are filtered on the basis of vicinity, availability and reliability. Unless a resource is unique or rare, our algorithm prefers to select locally available resources due to faster communication. Resource availability is maintained through Mean Time Between Availability (MTBA) and Mean Time Between Unavailability (MTBU) and a resource having high MTBA and less MTBU is selected. Reliability of a resource is calculated by maintaining information about successful and unsuccessful job executions and types of failures encountered. Resources having hardware failures are avoided. Each resource is assigned '1' or '0' for locality, availability and reliability by resource filtration module. Resource filtration matrix then identifies either to select a resource or otherwise. We devise a Genetic algorithm based Optimal Resource Identification (GORI) mechanism to pick and choose better resources for job execution. Jobs utilizing selected grid resources will generate more financial gains for the resource owners as these resources are optimal, highly available and more reliable than the rest of grid resources. Overall QoS will be enhanced due to resource selection on the basis of their optimality, availability and reliability.

Reactive FT orchestrator is equipped with two modules, i) failure predictor and ii) failure detector. Each grid resource running user jobs has failure predictor module installed and running for predicting or identifying hardware failure on the basis of temperature of the devices available in it. If the temperature of resource entities goes out of normal operating ranges, then failure predictor considers it a 'prediction failure' and reduces checkpoint intensity and updates failure detector about the change. If the temperature goes out of the acceptable ranges, then failure predictor considers it a 'hardware failure' and informs failure detector. Failure detector appropriately coordinates with proactive FT orchestrator and the job is moved to some other grid resource. Failure detector detects communication/link failures by sending ''ping liveness'' messages to those grid resources that are executing jobs. Reply not received from a resource within acceptable time frame is perceived as a 'network failure'. Upon identifying network failure, failure detector updates proactive FT orchestrator about the failure and restarts the job from the last saved checkpoint on other available grid resources. The performance of the proposed scheme is validated through extensive simulation experiments. Simulation results demonstrate the performance supremacy of the proposed scheme in terms of various performance metrics such as throughput, waiting and turnaround time, number of checkpoints and energy consumption.

To the best of our knowledge and literature review, none of the existing Fault Tolerant Scheduling Schemes (FTSS) consider resource selection based on vicinity and availability.

The major contributions of this work are as follows:

A. Design of a dynamic and adaptive fault tolerant scheduling strategy for computational grids that filters grid resources based on vicinity, availability and reliability.
B. Identification of optimal resource pool using 'Rank' selection through Genetic algorithm. Optimal resources are used in job execution for performance enhancement, reduced energy consumption, high throughput and more financial gains.
C. In order to identify and handle failure, following are proposed:
    1) Use of push and pull based models.
    2) Temperature based discovery of suspicious resources having more expectancy of failures through designed failure predictor.
    3) Discovery of link/communication failure through failure detector.
    4) Controlling checkpoint intensity on the basis of information received from device temperature.

Rest of the paper is organized as follows: related work is discussed in Section II; problem description is mentioned in Section III; Section IV presents proposed fault tolerant scheduling scheme. Experimental methodology is part of Section V; results and analysis are discussed in Section VI. Section VII concludes the paper.

II. RELATED WORK

There could be many reasons due to which a job submitted for execution in the grid can fail. Many researchers [2], [9], [10] have highlighted the types of errors, failures and faults expected in HPC environments. Reason for failures can be hardware, software, network, performance and perhaps many more that have not yet been conceived. Research carried out by [11] claims that hardware is the most important cause of failures in all HPC systems. Many fault tolerant proactive,

reactive and hybrid (proactive and reactive) approaches have been proposed [3], [7], [8], [10], [12]–[17] to cater and minimize the ill effects caused by the faults during the execution of the jobs in grids and other HPC environments. Careful selection of resources can help in reducing the probability of failures. Random selection of resources leads to increased frequency of failures causing reduction in throughput and performance.

A. REACTIVE FAULT TOLERANCE

Reactive or post-active fault tolerant approaches mostly rely on fault identification as the failure related action is taken after its detection. If a grid node is unable to respond due to communication failure, then retry or replication fault tolerant techniques can be applied after detection of the failure. More the time taken in detection of failure more the loss faced with respect to performance. Retry, replication, message logging and checkpointing are the techniques used in reactive fault tolerance.

An adaptive checkpointing strategy was presented in [18] for unreliable grid systems. Periodic checkpointing on stable and unstable resource is controlled for reducing checkpointing overheads. Proposed Last Failure Dependent Checkpointing (LFDC) algorithm adjusts starting checkpointing interval according to the behavior of resource and the execution time required by the task for generating a customized checkpointing frequency. Timestamp is stored for a resource failure and is used for controlling checkpointing intensity.

Message logging based protocol was presented by [19] to avoid complete restoration of a process in case of failure. Message logging protocol is executed at user level instead of library level through constructing it on User Level Failure Mitigation (ULFM). Placing a layer of fault tolerance over ULFM results improve portability with respect to faults. Recording complete communication statements in sender's message log without dividing them into implementation details further reduce fault tolerant overheads for collective communications.

A replication based fault tolerance technique was proposed by [20]. Proposed technique maintains fault index of resources based on fault history. Fault index value updates accordingly upon success or failure of task assigned on the resource. A job submitted for execution replicates on other backup resources also. A fault identified during execution of a job can get results and proceed by obtaining information from backup resources.

B. PROACTIVE FAULT TOLERANCE

Faults expected during job executions can be handled proactively. Resource failure can be identified using probability based techniques or through observing the behavior of resources during job execution. Failure history of resources can also be maintained for selection of reliable resources. Resource scheduling with respect to fault tolerance leads to Fault Tolerant Scheduling (FTS).

A fault tolerant Scheduling Indicator based Strategy (SIS) was presented by [21]. Resource selection is made on the basis of scheduling indicator that consists of response time and failure rate of grid resources. When a job requests resources, then resource is provided on the decision made by the indicator. The technique used for minimizing faults is to keep information about resource failure. When a resource is required, then select those resources that have less failure tendency.

A Multi constrained load balancing Fault Tolerant (MFT) scheduling for grid computing environment was proposed by [3]. The algorithm focuses on fault tolerance and load balancing of resources. Upon submitting a job for execution resource selection is made on the basis of initial failure rate, number of jobs submitted, successfully executed jobs and processing capability of the resource. A fault handler module maintains information about resource failures. Other modules include 'deadline control', 'load balancing', and 'budget control module'.

A fault tolerant job scheduling system for economy based grid environments was proposed by [22]. The proposed system maintains a fault index of every grid resource that updates dynamically upon successful or unsuccessful job completion. If a job completes execution within the defined time, it is considered a success and the fault index value of the resource is decreased. Similarly, if a job fails to complete execution within the allotted time, it is considered a failure and appropriately the value of the fault index of the resource is increased. Resource selection for the job is based on the value of the fault index. High fault index value represents high failure probability of that resource. Furthermore the intensity of checkpoints is also based on the value of the fault index. The proposed fault tolerant scheduling strategy is compared with time optimization heuristics for economy based grid environments.

C. HYBRID FAULT TOLERANCE

Hybrid fault tolerance uses combination of proactive and reactive approaches. Initially resources are selected using proactive methods and in case of failures reactive techniques are applied for appropriate handling of fault based situation.

A combination of proactive and reactive approaches (PRF) was used by [14] for handling faults in grid based environments. In proactive phase, the designed system selects suitable resources by identifying their reliability and current status. Benefits of proactive approach are that failure probabilities are reduced during the job execution and number of rescheduling are minimized. In reactive phase, failed jobs are started from the last saved checkpoint with minimized recovery time. The designed system calculates the value of every resource in the grid based on its reliability and status. Job performance time is also calculated on the basis of execution time, waiting time and transfer time. A checkpoint manager maintains checkpoint intervals during job execution.

Hybrid fault tolerance (HFT) technique for grid computing was presented by [23]. The proposed work combines

good features of workflow level and task level fault tolerant techniques and eliminates demerits of both. Retry, alternate resource, checkpoint and replication are the task level fault tolerance techniques and are considered as task level recovery techniques. Workflow level fault tolerance techniques change execution flow upon failures. Alternate task, redundancy, user defined exception handling and rescue workflow are the types of workflow level fault tolerance. Hybrid techniques discussed in the paper are alternate task with checkpoint and alternate task with retry. Performance metrics used in the paper are throughput, turnaround time, waiting time and transmission delay.

Most of the work related to fault tolerance reveals that still there are factors like i) resource location, ii) availability time and iii) unavailability time of resources; requiring attention and consideration of researchers. More focus on fault tolerance or a bulky fault tolerant architecture also affects performance. A good FTSS might suffer performance degradation. Fault tolerant scheduling system must not overlook performance related issues. Most of the fault tolerant scheduling techniques pay more attention on reliable resource identification and neglect issues like resource failure detection. Failure identification and detection is an important area from fault tolerance point of view that could help in improving grid dependability. A good failure detection mechanism in addition to fault tolerant scheduling system could be a good combination and must be considered for improved dependability and reliability. The motivation is of establishing a FTSS that could improve performance and detect failures in addition to performing fault tolerant scheduling duties.

III. PROBLEM DESCRIPTION

Most of the discussed fault tolerant scheduling techniques emphasize on resource selection on the basis of maintaining historical data. Maintaining values of fault index and scheduling indicator and selecting resource on these numbers cover only one aspect of the problem. There are some other very important factors that have been neglected in the design of fault tolerant HPC systems and specially grids. Throughput, waiting time and turnaround times cannot be improved merely by focusing on resource behavior. Availability of a resource is an important consideration for ensuring reliability and QoS of a grid. Grid consists of locally and remotely connected resources that can join or leave it any time. A resource connected with grid for a long time is considered to be highly available.

Consider a resource whose availability time in the grid is less than its unavailability time. Most of the fault tolerant scheduling models will select this resource for job execution despite having less availability time, due to its high success ratio. A resource having less availability time cannot produce high throughput. For improving grid performance, we must take into account the factor of resource availability [2]. In most FTSS, this important factor is overlooked.

Location of resource is another important factor that has been neglected during resource selection. Many FTSS have claimed improvement in reducing communication and network delays without considering the location of the selected resources. Resource selection on the basis of location must be considered for improving the performance of grid. Resources located nearby or in close vicinity will always communicate faster than a remotely connected resource and this property has not been exploited for resource selection and performance improvement in grid and other HPC environments.

Another problem identified during resource selection in most of the fault tolerant scheduling systems is their inability to identify optimal resources. Most of the fault tolerant systems do not consider resource optimality during selections. Consider a set of 100 grid resources out of which 5 resources are very fast and capable from processing and storage point of view and the failure index or scheduling indicator of all grid resources is equal. Merely considering failure index or scheduling indicator will have only 5% probability of selecting those optimal resources. Most of the fault tolerant schedulers will not select computationally faster resources due to the consideration of resource selection on the basis of failure index or scheduling indicator, due to which performance enhancement and improved QoS will not be possible. The design of fault tolerant schedulers should consider resource selection on the basis of availability, location and optimality in addition to reliability.

IV. DYNAMIC AND ADAPTIVE FAULT TOLERANT SCHEDULING SYSTEM

In this section we explain the proposed system that uses dynamic and adaptive fault tolerant scheduling scheme. Fig. 1 shows the interaction of different grid components from user job submission to job completion. Two main components of the FTSS are i) proactive and ii) reactive fault tolerant orchestrator.

Proactive FT orchestrator further uses two main components, i.e. i) resource filtration and ii) Genetic algorithm based Optimal Resource Identification (GORI). Reactive FT orchestrator submits the jobs for execution on the grid and also contains two important components i) failure predictor and ii) failure detector. Fig. 2 shows the main components of the proposed model. Every grid resource executing user jobs has failure predictor component that interacts with failure detector upon noticeable change in the temperature of resource entities. Failure predictor passes two types of failure related information to failure detector, i) prediction failure and ii) hardware failure. Failure detector periodically sends ping messages to grid resources executing user jobs and reply not received within specified time is considered network failure.

A. GRID USER

Grid users request computing resources through grid web portal. Job sent by the user is received by grid scheduler. Grid scheduler contacts Grid Information Server (GIS) that contains information about grid resources. GIS provides grid resource information to grid scheduler. Grid scheduler

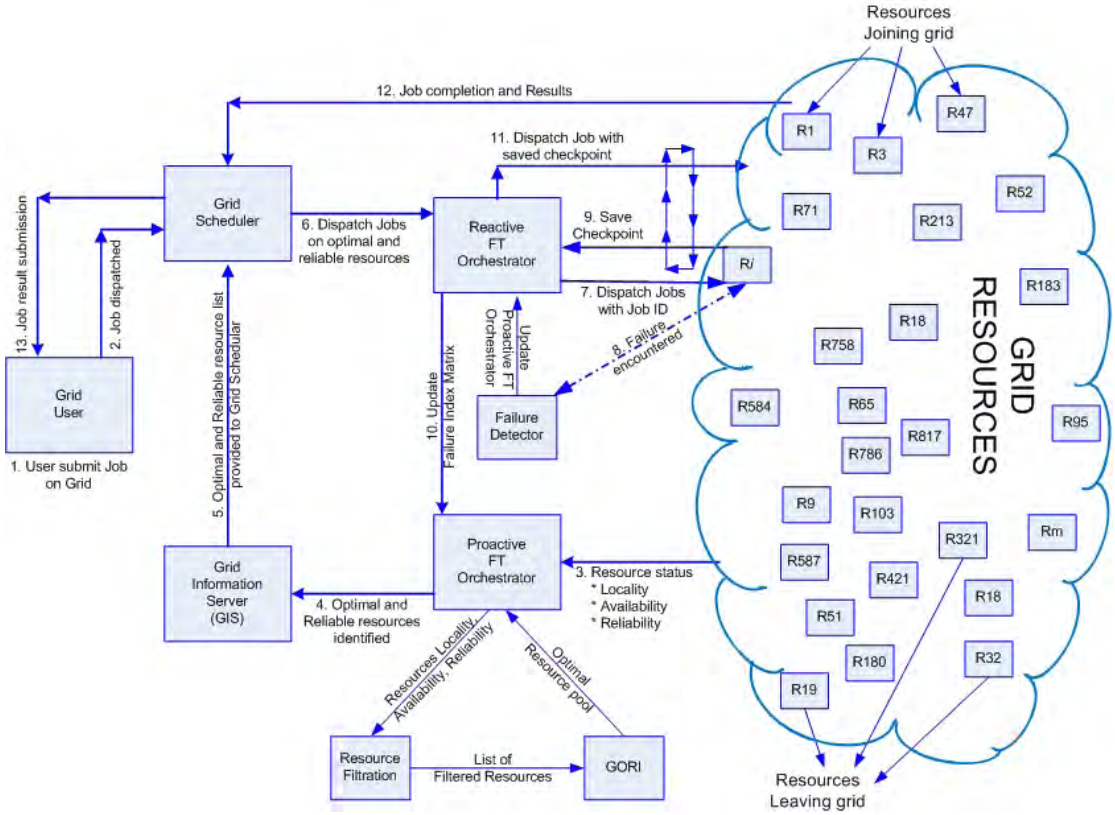

FIGURE 1. Dynamic and adaptive fault tolerant scheduling.

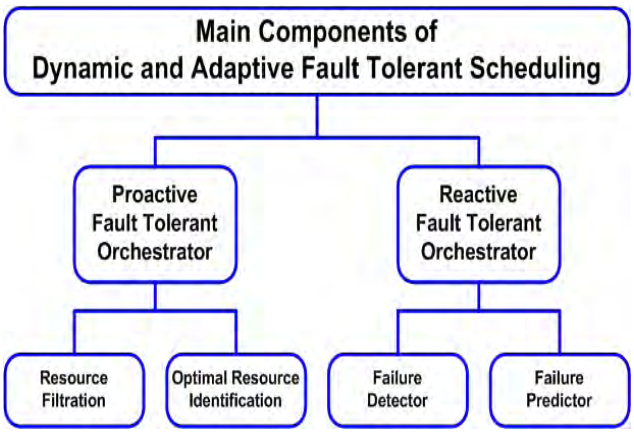

FIGURE 2. Components of dynamic and adaptive fault tolerant scheduling.

dispatch jobs on grid resources. After completion of the job, results are returned to the user.

B. GRID SCHEDULER

In the proposed model, jobs are dispatched on grid resources by reactive FT orchestrator. Scheduler in the proposed model before dispatching the job on grid resources contacts GIS that obtains the list of reliable and optimal resources from proactive FT orchestrator. Jobs are dispatched on identified optimal resources. Reactive FT orchestrator shifts the jobs to other available resources for hardware and network failures, and in case of prediction failure, checkpoint intensity is appropriately adjusted. Information regarding failures is shared with proactive FT orchestrator for reliability enhancement of resources. After completion of the job results are submitted to the user.

Components of proactive and reactive FT orchestrator are shown in Fig. 2. In proactive FT orchestrator we have devised a resource filtration mechanism in which the resources are filtered on the basis of location, availability and reliability. Resource filtration algorithm filters grid resources on the basis of their location (local or remote), resource availability time (time when resource become part of the grid), and resource reliability based on (number of failures, failure type and failure intensity). Failure intensity is generated by failure index matrix whereas the information in the matrix regarding type and number of failures is updated through failure detector.

Factors considered for resource filtration are shown in Fig. 3. After resource filtration process, filtered resources are passed to genetic algorithm that identifies and produces optimal resource pool. Fitness function in the genetic algorithm finds the optimal resources according to the weights assigned to hardware configurations. Rank selection technique is used for finding the fittest resources.

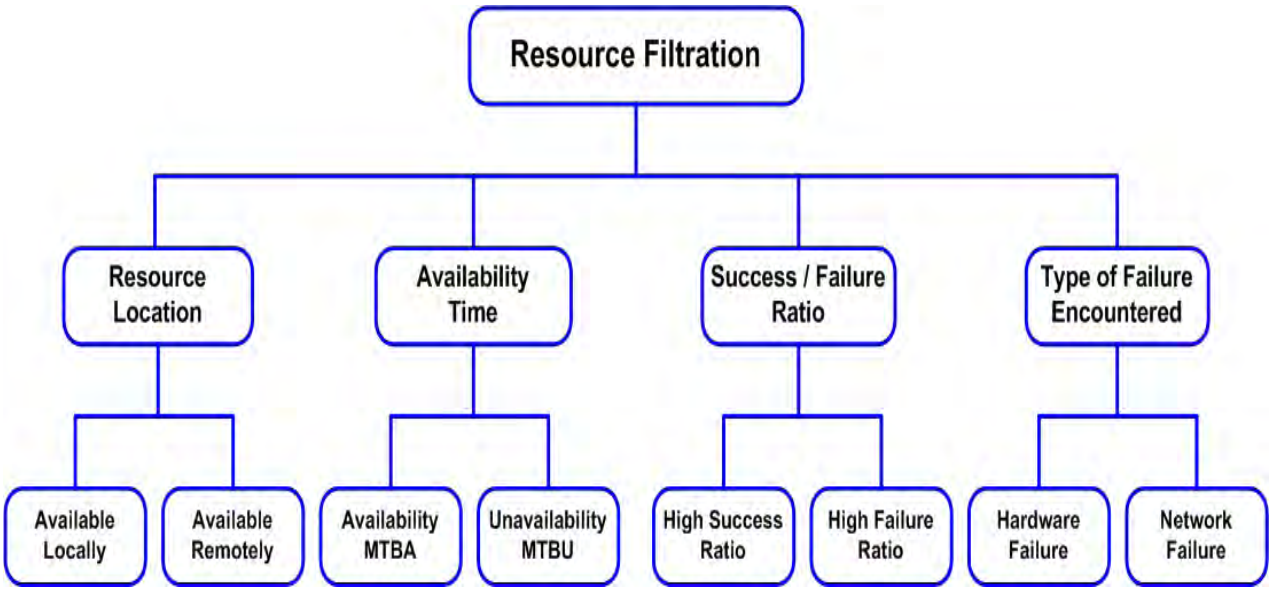

FIGURE 3. Resource filtration criteria's

Reactive FT orchestrator submits the job on the resources identified through proactive FT orchestrator. Failure detector and failure predictor use pull and push based mechanism for detecting and predicting network and hardware failures. Reactive FT orchestrator updates proactive FT orchestrator for successfully executed jobs or resource failures. Results are submitted to the user after the completion of the job. Detailed working of the components of proactive and reactive FT orchestrators is described.

C. RESOURCE LOCATION

Resources are the actual strength of a grid. Resources available in grid are i) locally available and ii) remotely available. Locally available resources are part of the grid and work under the authority of the grid maintainer. Remotely available resources are connected and available for use through communication links and are bound to work under the policies of the domains of which they are a part. A change in policy or configuration of the network of remotely connected resource can raise questions about the availability and accessibility of that resource. If a remotely accessible grid resource is very unique then we have no choice of selection other than this because of its uniqueness, but if it is like other commonly available resources, than its locality must be considered into account for appropriate ranking. Things can further be complicated if communication link goes down or if the communication speed reduces due to various factors not under the control of grid maintainer. So, locally available resources would always have an edge with respect to fault tolerance over remotely accessible resources. To our knowledge, this issue has never been exploited in resource ranking of grids with respect to fault tolerance. Weightages assigned to locally and remotely connected resources are mentioned in Table 1.

TABLE 1. Location ranking.

Resource location	Weightage
Locally connected	1
Remotely connected	0

List of notations used in the algorithms is mentioned in Table 2:

Algorithm 1 describes the selection of grid resource based on locality:

Algorithm 1 Locality Identification of R_i

Input: All grid resources
Output: Locally available grid resources
Begin
1: **for** $R_i = 1$ *to* m **do**
2: **if** $(R_i \in LT_{lcr})$
3: $LT_{lcr} \leftarrow 1$
4: **else**
5: $LT_{lcr} \leftarrow 0$
6: **end if**
7: **end for**

D. RESOURCE AVAILABILITY TIME

Grid resources are part of various geographical domains that can join or leave any time [14]. A resource that has more tendency to leave grid without any prior information is an important point for consideration if we rate grid resources. To the best of our knowledge this type of resource behavior has not been exploited previously for resource ranking with respect to fault tolerance. Availability time of a resource is the time since it joined the grid. A resource connected with

TABLE 2. List of notations used in algorithms.

Notation	Definition
J_f	Number of jobs available in the system from 1 to f
R_m	Number of resources available in the system from 1 to m
Ri	A resource in the system
et_i	Execution time of resource *ri*
RL_{ri}	Reliability of resource *ri*
AT_{ri}	Availability time of resource *ri*
UT_{ri}	Unavailability time of resource *ri*
AL_{ri}	Availability of resource *ri*
LT_{lcr}	Location of resource *lcr* (Locally Connected Resource)
LT_{rcr}	Location of resource *rcr* (Remotely Connected Resource)
J_{rn}	Number of resources required for executing a job
MS_{jt}	Makespan or execution time of a job
FR	List of filtered resources from r1, r2, r3, . . . , *rn*
UR	List of unfiltered resources from r1, r2, r3, . . . , *rm*
RI_{ri}	Reliability identification of resource *ri*
TJ_{ri}	Total number of jobs executed on resource *ri*
SJ_{ri}	Successfully completed jobs on resource *ri*
FJ_{ri}	Failed jobs on resource *ri*
Ft_{rin}	Failure type 'network' on resource *ri*
Ft_{rip}	Failure type 'prediction' on resource *ri*
Ft_{rih}	Failure type 'hardware' on resource *ri*
Fi_{rin}	Failure intensity of failure type 'network' on resource *ri*
Fi_{rip}	Failure intensity of failure type 'prediction' on resource *ri*
Fi_{rih}	Failure intensity of failure type 'hardware' on resource *ri*
FI_{rinph}	Failure index of all types of failures on resource *ri*
FS_{ri}	Filtration score of resource *ri*
tv	Threshold value for fitness function
P	Number of candidate solutions in a generation
n	Size of set P
c_p	Crossover point
c_r	Rate of offspring taking part in crossover
m_r	Percentage of mutation or mutation rate
J_{rn}	Number of resources required by user for executing job
s_l	Length of the string
g_i	i^{th} offspring in P
PS	Population obtained after crossover
R_{nr}	Required number of resources
OR_p	Optimal resource pool, e.g. output of genetic algorithm

the grid for a longer time will have high availability time. We can calculate availability and unavailability time of a resource when it joins the grid for the first time or when it rejoins the grid after leaving. Equation (1) shows the availability time of a resource "i".

$$AT\sum_{ri}^{1\to m} \leftarrow AT1_{ri} + AT2_{ri} + ... + ATm_{ri} \qquad (1)$$

Time for which a resource was not available is called unavailability time. A resource that leaves the grid more often for a longer period of time will have high unavailability or less availability. Unavailability time of a resource is the sum of times for which it was not available. Equation (2) shows the unavailability time calculation of a resource "i".

$$UT\sum_{ri}^{1\to m} \leftarrow UT1_{ri} + UT2_{ri} + ... + UTm_{ri} \qquad (2)$$

Mean time between availability of a resource is the sum of all available times divided by number of times it joined the grid. MTBA calculation of a resource "i" is shown in equation (3).

$$\bar{AT}_{ri} \leftarrow (AT1_{ri} + AT2_{ri} + ... + ATm_{ri})/m \qquad (3)$$

MTBA is similar to Mean Time Between Failure (MTBF), except the later provides time between two failures [2] and for reliable machines it should be high. Similarly, MTBA is a time for a resource in which it was available for use in the grid. Again for reliable resource MTBA should be high and for unreliable resource that keeps on joining the grid for a shorter period of time, it would be low. Mean time between unavailability of a resource is the sum of all unavailable times divided by number of times it left the grid. MTBU calculation of a resource "i" is shown in equation (4).

$$\bar{UT}_{ri} \leftarrow (UT1_{ri} + UT2_{ri} + ... + UTm_{ri})/m \qquad (4)$$

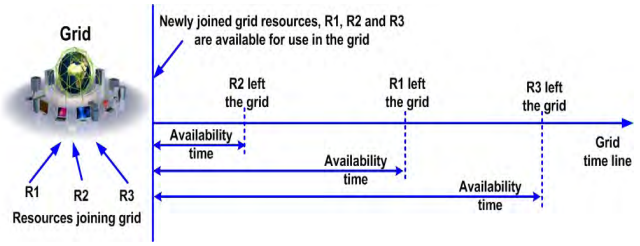

FIGURE 4. Availability time of grid resources.

Designed fault tolerant orchestrator keeps timing related information of all the resources that have joined the grid. Resources connected with the grid for the longest period of time will be preferred for selection as compared to the newly joined resources. Fig. 4 shows the availability times of resources R1, R2 and R3.

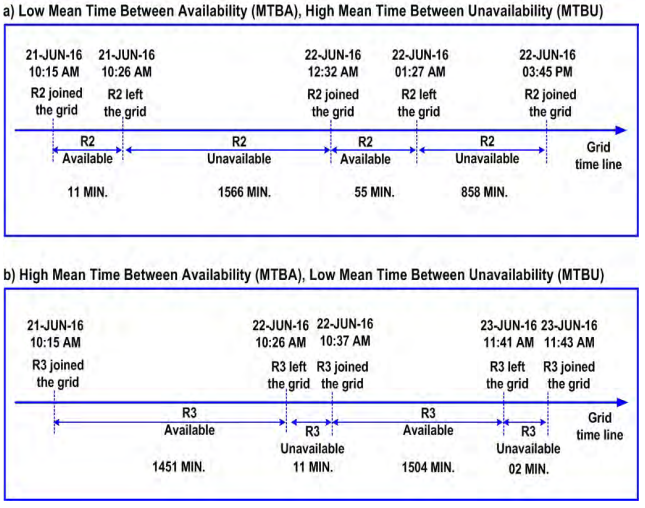

FIGURE 5. MTBA and MTBU of resources joining and leaving grid.

MTBA for preferred resources should be high. MTBU for preferred and reliable resource would be low. High MTBA and low MTBU is a QoS parameter. Fig. 5 shows the availability and unavailability times of resource R2 and R3 with respect to joining and leaving the grid. It can be seen in fig. 5(a), that resource R2 is leaving the grid quickly and for more time, due to which its calculated MTBU would be high and is not an ideal or desirable resource from stability point of view.

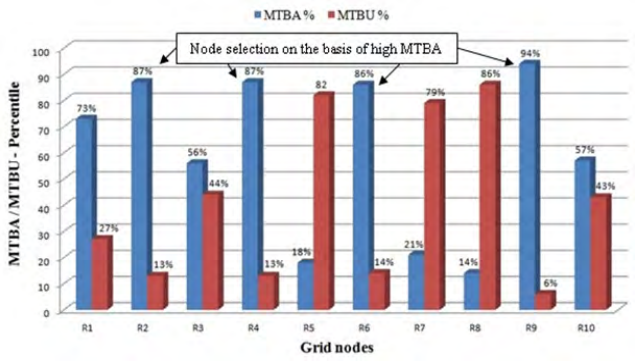

FIGURE 6. MTBA and MTBU percentages of grid nodes.

TABLE 3. Availability ranking.

Resource availability	Weightage
High	1
Low	0

In fig. 5(b), resource R3 is shown in grid time line with respect to joining and leaving the grid. It can be seen in the time line that resource R3 leaves the grid for a smaller amount of time and remains connected with the grid for a longer period of time. Connectivity time of resource R3 on the grid time line is mentioned and by calculating the mean of availability we obtain a higher MTBA as compared to MTBU. High MTBA and low MTBU are desired for reliable resources. An ideal resource selection for computation in grid will be the one having high MTBA and low MTBU as shown in fig. 6. Availability of individual resource will be ranked according to the availability ranking shown in Table 3.

Resource selection on the basis of availability is described in Algorithm 2:

Algorithm 2 Availability Identification of R_i

Input: Availability and Unavailability time of all grid resources
Output: Resources having High MTBA
Begin
for $R_i = 1$ *to* m **do**
  $AT\sum_{ri}^{1\to m} \leftarrow AT1_{ri} + AT2_{ri} + ... + ATm_{ri}$
  $UT\sum_{ri}^{1\to m} \leftarrow UT1_{ri} + UT2_{ri} + ... + UTm_{ri}$
  $\bar{AT}_{ri} \leftarrow (AT1_{ri} + AT2_{ri} + ... + ATm_{ri})/\mathrm{m}$
  $\bar{UT}_{ri} \leftarrow (UT1_{ri} + UT2_{ri} + ... + UTm_{ri})/\mathrm{m}$
  if $(\bar{AT}_{ri} > \bar{UT}_{ri})$
    $AT_{ri} \leftarrow 1$
  else
    $AT_{ri} \leftarrow 0$;
  end if
end for

E. SUCCESS/FAILURE RATIO OF RESOURCES

Many researchers have used the approach of identifying the reliable resource based on historical data. It is relatively easy to maintain and identify resources having less tendency of failure, by maintaining the count of successful execution. Our algorithm stores history based information of the jobs executed on the grid and their behavior during execution in failure index matrix Table 4.

Nodes having the highest failure index will not be selected for execution. Failure index is calculated on the basis of factors mentioned in Table 4. Some of the values in Table 4; like i) jobs submitted for execution on a resource, ii) jobs completed successfully, iii) failures encountered during execution, and iv) types of failures are received through failure detector component.

As the grid resource starts executing a job, the algorithm starts updating the values for identification of reliable resources based on historical information maintained in failure index matrix. Methodology of filling the values in failure index matrix is shown in Table 4. Initially, failure index of all nodes will be started from 0 and will be incremented by 1 by failure detector component upon successful completion or failure encountered. Our proposed system caters three types of failures i.e., i) network failures, ii) prediction failures (information obtained from failure predictor based on monitoring temperature of hardware devices), and iii) hardware failure. Data updated in failure index matrix for some nodes is shown in Table 4. Jobs submitted on node ID 'R6' and 'R7' are 20 and both encountered 4 failures each. Depending on the type of failure, node ID 'R7' has the highest failure index value of 65, whereas node ID 'R6' has the lowest failure index value of 5. Similarly, total jobs submitted on node ID 'R1' is 15 out of which 9 executed successfully and 6 of them encountered failures, yielding failure index value of 10. Types of failures are maintained in the matrix for identification and calculation of failure index.

Three types of failures identified by our proposed system have appropriate failure intensities. Individual failure intensity is calculated using weights assigned for each failure type and the number of failures encountered. Value of failure index is generated by adding all individual failure intensities. Most of the failures encountered in distributed environments are due to hardware [11] and because of this very fact it has appropriate importance in failure intensity for calculating failure index. Failure index increases substantially when a hardware failure is encountered. Hardware failure has the highest failure index and algorithm will avoid selecting the nodes that have encountered hardware failures.

Table 5 is extracted from Table 4 and is in ascending order of failure index values. It can be seen in Table 5 that nodes having low failure index will be selected as 'reliable' nodes. Reliable nodes get '1' weightage. High failure index is calculated by the system due to hardware failure. Nodes having hardware failures will be declared as unreliable and appropriately will be assigned 0 weightage as shown in Table 5 as well as in Table 6. The value of failure index generated by failure index matrix is highest where hardware failures are encountered.

Selection of grid resources on the basis of reliability is elucidated in Algorithm 3:

TABLE 4. Failure index matrix.

Node ID	Total Jobs Submitted	Completed	Failed	Failure Type 1. Network 2. Predicted 3. Hardware			Failure Intensity with assigned weights Network=1 Predicted=2 Hardware=3			Failure Index
				N/W	Predicted	H/W	N/W	Predicted	H/W	
R1	15	9	6	2	4	0	1x2=2	2x4=8	3x0x15=0	2+8+0=10
R2	15	8	7	1	5	1	1x1=1	2x5=10	3x1x15=45	1+10+45=56
R3	20	10	10	6	4	0	1x6=6	2x4=8	3x0x20=0	6+8+0=14
R4	20	16	4	2	2	0	1x2=2	2x2=4	3x0x20=0	2+4+0=6
R5	20	4	16	12	4	0	1x12=12	2x4=8	3x0x20=0	12+8+0=20
R6	20	16	4	3	1	0	1x3=3	2x1=2	3x0x20=0	3+2+0=5
R7	20	16	4	1	2	1	1x1=1	2x2=4	3x1x20=60	1+4+60=65

TABLE 5. Reliability calculation table sorted on failure index value.

Node ID	Total Jobs Submitted	Completed	Failed	Failure Type 1. Network 2. Predicted 3. Hardware			Failure Index (FI)	Reliability Status based on FI and H/W Failures.	Weightage assigned to nodes
				N/W	Predicted	H/W			
R6	20	16	4	3	1	0	5	Reliable	1
R4	20	16	4	2	2	0	6	Reliable	1
R1	15	9	6	2	4	0	10	Reliable	1
R3	20	10	10	6	4	0	14	Reliable	1
R5	20	4	16	12	4	0	20	Reliable	1
R2	15	8	7	1	5	1	56	Unreliable	0
R7	20	16	4	1	2	1	65	Unreliable	0

TABLE 6. Reliability ranking.

Resource reliability	Weightage
Minimum failure index with NO H/W Hardware failures	1 0

TABLE 7. Resource filtration matrix.

Combination	Location	Availability	Reliability	Filtration Matrix Score
(a)	1	1	1	1
(b)	1	1	0	0
(c)	1	0	1	1
(d)	1	0	0	0
(e)	0	1	1	1
(f)	0	1	0	0
(g)	0	0	1	0
(h)	0	0	0	0

F. RESOURCE FILTRATION

Resources identified on the basis of vicinity (Table 1), availability (Table 3) and reliability (Table 6) are passed to resource filtration algorithm that filters resources through filtration matrix rules. Combinations shown in resource filtration matrix (Table 7) generate matrix score '1' or '0' based on location, availability and reliability of grid resources. If reliability is '0' then the values of location and availability are ignored and '0' score is assigned to that node which disqualifies it from genetic algorithm input list. Any two or more combinations of '1' produce '1' for filtration matrix score which allows the genetic algorithm to select that node as input for the identification of optimal resource. Similarly, any two or more combinations of '0' generate '0' for filtration matrix which drops that particular node from input list of genetic algorithm for optimal resource selection. Resources selected or rejected through the filtration process are shown in fig. 7.

Algorithm 4 filters resources based on locality, availability and reliability.

G. INTERACTION BETWEEN COMPONENTS OF PROACTIVE FT ORCHESTRATOR

Filtered resource list is passed to GORI, the second component of proactive FT orchestrator. As grid resources vary in capacity or capability from each other, GORI identifies resources with respect to their capability. GORI will generate optimal pool of resources having i) high processing capabilities, ii) enhanced memory, iii) more storage

Algorithm 3 Reliability Identification of R_i

Input: Jobs completed, Jobs failed, types of failures
Output: Reliable grid resources
Begin
1: Receive SJ_{ri}, FJ_{ri}, Ft_{ri} from '*Failure Detector*'
2: Update '*Failure Index Matrix*' for R_i
3: **for** $R_i = 1$ *to* f **do**
4: $SJ\sum_{ri}^{x=1\to f} \leftarrow SJ1_{ri} + SJ2_{ri} + SJ3_{ri}.... + SJf_{ri}$
5: $FJ\sum_{ri}^{x=1\to f} \leftarrow FJ1_{ri} + FJ2_{ri} + FJ3_{ri}.... + FJf_{ri}$
6: $TJ_{ri} \leftarrow SJ\sum_{ri}^{x=1\to f} + FJ\sum_{ri}^{x=1\to f}$
7: $SJ_{ri} \leftarrow TJ_{ri} - FJ\sum_{ri}^{x=1\to f}$
8: $FJ_{ri} \leftarrow TJ_{ri} - SJ\sum_{ri}^{x=1\to f}$
9: $Ft\sum_{rin}^{n=1\to x} \leftarrow Ft_{rin1} + Ft_{rin2} + Ft_{rin3}.... + Ft_{rinx}$
10: $Ft\sum_{rip}^{p=1\to x} \leftarrow Ft_{rip1} + Ft_{rip2} + Ft_{rip3}.... + Ft_{ripx}$
11: $Ft\sum_{rih}^{h=1\to x} \leftarrow Ft_{rih1} + Ft_{rih2} + Ft_{rih3}.... + Ft_{rihx}$
12: $Fi_{rin} \leftarrow Ft\sum_{rin}^{n=1\to x} * 1$
13: $Fi_{rip} \leftarrow Ft\sum_{rip}^{p=1\to x} * 2$
14: $Fi_{rih} \leftarrow Ft\sum_{rih}^{h=1\to x} * TJ_{ri} * 3$
15: $FI_{ri} \leftarrow Fi_{rin} + Fi_{rip} + Fi_{rih}$
16: **if** $\left(Ft\sum_{rih}^{h=1\to x} \geq 1\right)$
17: $RI_{ri} \leftarrow 0$
18: **else**
19: $RI_{ri} \leftarrow 1$
20: **end if**
21: **end for**

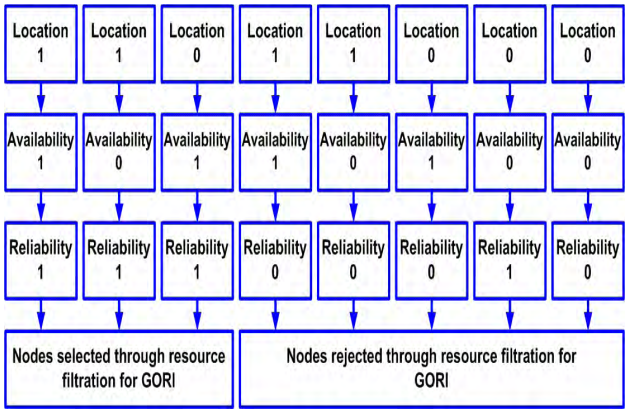

FIGURE 7. Resource filtration based on location, availability and reliability.

capacity and iv) fast communication link. Benefits gained by selecting resources through GORI are; i) job execution time will be minimized, ii) throughput will be increased, iii) resource selection will be faster, iv) more financial gain, v) less energy consumption and vi) overall quality of service will be enhanced. Jobs having execution requirements under strict time constraints will be able to complete within allotted deadline period and will not encounter QoS failure. Fig. 8 shows the interaction between two components of proactive FT orchestrator, i) resource filtration and ii) GORI.

Algorithm 4 Resource Filtration

Input: Output of algorithm 1, 2 and 3
Output: Filtered resources based on Locality, Availability and Reliability
Begin
1: **for** $R_i = 1$ *to* k **do**
2: **if** $(RI_{ri} \leftarrow 0)$
3: $FS_{ri} \leftarrow 0$
4: goto 1
5: **end if**
6: **if** $(LT_{ri} == 1 \&\& AT_{ri} == 1)$ **or**
7: **if** $(LT_{ri} == 1 \&\& RI_{ri} == 1)$ **or**
8: **if** $(AT_{ri} == 1 \&\& RI_{ri} == 1)$
9: $FS_{ri} \leftarrow 1;$
10: **else**
11: $FS_{ri} \leftarrow 0;$
12: **end if**
13: **end for**

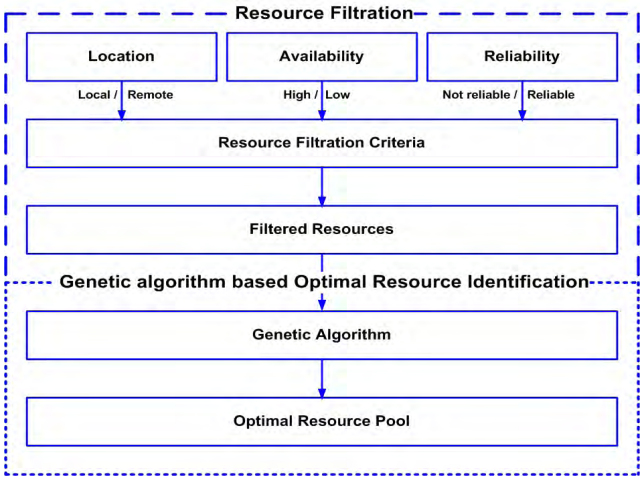

FIGURE 8. Interaction between components of proactive FT orchestrator.

H. GENETIC ALGORITHM BASED OPTIMAL RESOURCE IDENTIFICATION (GORI)

Here we give the details of genetic algorithm through which optimal resource pool of grid resources will be generated. Genetic algorithm optimal resource identifier will provide the list of optimal grid resources. At the start, Genetic Algorithm (GA) generates initial population 'P' of size 'n' using a function named 'initialization'. Initially it populates an empty set 'P' to a size 'n' making sure that no two offspring's are same. After 'initialization', 'evaluation' function computes fitness value against each of the generated offspring using the following function:

$$\text{Fitness}\ (g_i) = 0.45w + 0.30x + 0.15y + 0.1z$$

Where variable 'w', 'x', 'y', and 'z' are resource constraints and their description is as follows:

'w' represents the capability of CPU and the weightage assigned to it is 0.45 as it is most important resource of

a system. 'x' represent the capability of memory and the weightage assigned to it is 0.30 according to its importance. 'y' represents storage capacity and the weight assigned to it is 0.15. 'z' represents communication link speed and has the weightage of 0.1. The sum of weights of all resource constraints is equal to 1, e.g. $0.45 + 0.30 + 0.15 + 0.1 = 1$.

TABLE 8. Resource placement on the basis of its strength.

Resource ID	Resource Type	Resource placement category		
		Low = 1	Medium = 2	High = 3
'w'	CPU speed	< 2 GHz	≥ 2GHz and ≤ 4 GHz	> 4 GHz
'x'	Memory capacity	< 1 GB	≥ 1 GB and ≤ 4 GB	> 4 GB
'y'	Storage capacity	< 500 GB	≥ 500 GB and ≤ 1 TB	> 1 TB
'z'	Comm. speed	≥ 10 Mbps	≥ 100 Mbps	≥ 1 Gbps

TABLE 9. Resource capacity/strength of nodes.

Node ID	CPU Speed	Memory Capacity	Storage	Communication Link Speed
R1	3.00 GHz	2 GB	1 TB	100 Mbps
R2	1.56 GHz	512 MB	500 GB	10 Mbps
R3	4.56 GHz	4 GB	500 GB	100 Mbps
R4	5.42 GHz	8 GB	2 TB	1000 Mbps
R5	5.06 GHz	4 GB	500 GB	1000 Mbps
R6	5.24 GHz	8 GB	4 TB	1000 Mbps
R7	2.53 GHz	2 GB	500 GB	100 Mbps

Each resource w, x, y and z according to its capability is placed in low, medium and high category. Table 8 shows resource placement category according to its capability or strength and the values assigned to low, medium and high categories. Table 9 shows resource capacity of some nodes.

TABLE 10. Values assigned to reseource capacity of grid nodes.

Node ID	CPU Speed	Memory Capacity	Storage	Communication Link Speed
R1	2	2	2	2
R2	1	1	2	1
R3	3	2	2	2
R4	3	3	3	3
R5	3	2	2	3
R6	3	3	3	3
R7	2	2	2	2

Table 10 shows the values assigned according to resource capacity/strength of grid nodes. 1, 2 and 3 correspond to strengths of low, medium and high.

After generating the fitness values the algorithm iteratively performs three stochastic operators until the fitness value for the selected offspring becomes greater than the given threshold value 'tv'. The stochastic operators are 'select', 'crossover' and 'mutation'. Table 11 shows the fitness values of some sample strings generated by genetic algorithm according to the strength of the resource constraints of grid nodes.

TABLE 11. Fitness values according to resource constraints.

Node ID	CPU Speed 'w'	Memory Capacity 'x'	Storage Capacity 'y'	Comm. Speed 'z'	Fitness Value $\sum(w,x,y,z)$
R1	0.45 * 2 = 0.90	0.30 * 2 = 0.60	0.15 * 2 = 0.30	0.1 * 2 = 0.20	2.00
R2	0.45 * 1 = 0.45	0.30 * 1 = 0.30	0.15 * 2 = 0.30	0.1 * 1 = 0.10	1.15
R3	0.45 * 3 = 1.35	0.30 * 2 = 0.60	0.15 * 2 = 0.30	0.1 * 2 = 0.20	2.45
R4	0.45 * 3 = 1.35	0.30 * 3 = 0.90	0.15 * 3 = 0.45	0.1 * 3 = 0.30	3.00
R5	0.45 * 3 = 1.35	0.30 * 2 = 0.60	0.15 * 2 = 0.30	0.1 * 3 = 0.30	2.55
R6	0.45 * 3 = 1.35	0.30 * 3 = 0.90	0.15 * 3 = 0.45	0.1 * 3 = 0.30	3.00
R7	0.45 * 2 = 0.90	0.30 * 2 = 0.60	0.15 * 2 = 0.30	0.1 * 2 = 0.20	2.00

1) STOCHASTIC OPERATION - SELECT

There are two strategies for selection of the offspring's which will participate in next generation. One is 'rank selection' i.e. pick up the best every time; and the other is 'roulette wheel selection' which randomly selects the participants for next generation. The stochastic operation for selection of offspring's used in our algorithm is 'Rank selection' strategy. By having select operator, we make sure that stronger (with more fitness value) strings will participate more in the next generation while the weaker ones will die off.

2) STOCHASTIC OPERATION - CROSSOVER

Next we apply crossover among the string structures. For crossover, the input 'c' is used which is the percentage of crossover to be applied in set 'P'. Crossover takes place in two parts. In the first part, the strings to be combined are chosen at random. In the second part, crossover point between the chosen pairs is also chosen at random between 1 to 's_l', where 's_l' is the length of the string. Crossover takes place by swapping partial bits of the selected pairs across the crossover point. Crossover generates bulk amount of new information in the new generation.

3) STOCHASTIC OPERATION - MUTATION

After 'crossover', 'mutation' is applied which is a process of randomly inverting a bit's position with appropriate percentage of mutation. The percentage of mutation applied in our algorithm is 'm_r'. Genetic algorithm will provide the number of optimal resources R_{nr} as required by the user. Furthermore, input 'P' in genetic algorithm is not constant and keeps on increasing/decreasing due to the reason that resources in the grid keep joining and leaving, and our designed resource filtration algorithm keeps updating the list of available and reliable resources. Once the genetic algorithm passes the required threshold value 'tv', the algorithm compares the ranks of offspring's in 'PS' with the required number of resource 'R_{nr}'. If the rank lies between the range 1 to 'R_{nr}', the algorithm returns the corresponding solution to optimal resource pool.

Here we present the Genetic algorithm based Optimal Resource Identification algorithm.

Algorithm 5 Genetic Algorithm Based Optimal Resource Identification (GORI) [6]

Input: Output of algorithm 4
Output: Optimal resource pool
Begin
initialization();
evaluation();
while (max(fitness(g_i in P)) > tv)
  $PS = \Phi$;
  select(); // rank selection
  crossover();
  mutation();
  $P = PS$;
  evaluation();
end while
for i = 1 to P **do**
  if (Rank(g_i in P) == member(1..R_{nr}))
    send g_i to optimal resource pool;
  end if
end for

Algorithm 5 (GORI) - Initialization Function [6]

start initialization()
  while ($|g_i| \leq P$)
    for j=1 to $\leq n$ **do**
      generate new g_i using r_j in R_m;
    end for
    if (g_i not in P)
      add g_i to P;
  end if
end initialization()

Algorithm 5 (GORI) - Evaluation Function [6]

start evaluation()
  for j=1 to $\leq P$ **do**
    fitness (g_i) = $\sum(0.45w, 0.30x, 0.15y, 0.1z)$;
  end for
end evaluation()

I. FAILURE PREDICTOR

Failure predictor module resides on all grid resources and interacts with reactive FT orchestrator. Resources that are involved in the process of job execution effectively use the proposed prediction services, as it is of no use to run failure predictor module on those grid resources that are not involved in processing of jobs. The designed failure predictor module uses temperature of the devices for prediction of failures. Designed technique focuses on hardware based failures as their expectancy and devastation is more

Algorithm 5 (GORI) - Select Function [6]

start select()
  for i=1 to $\leq P$ **do**
    if (Rank (g_i) <= (1-c) P)
      add g_i to PS ;
    end if
  end for
end select()

Algorithm 5 (GORI) - Crossover Function [6]

start crossover()
  for i=1 to $\leq C.P$ **do**
    select (g_m, g_n) pair from PS at random;
    find crossover point cp ($0 < c_p < s_l$) at random;
  end for
  for k=1 to $\leq c_r$ **do**
    add r_k from g_m to g_n;
    add r_k from g_n to g_m;
  end for
  for k = c_{r+1} to $k \leq J_{rn}$ **do**
    add r_k from g_m to g_n;
    add r_k from g_n to g_m;
  end for
end crossover()

Algorithm 5 (GORI) - Mutation Function [6]

start mutation()
  for i=1 to I $\leq m_r.P$ **do**
    select g_m from PS at random
    for j=1 to j $\leq J_{rm}$ **do**
      if (Random (1, 0) == 1)
        replace r_j by r_k ($r_{j \neq r_k}$; r_k in P);
      end if
    end for
  end for
end mutation()

in all HPC systems than of other types of failures [11]. Researchers have used the idea of hardware failure prediction on the basis of temperature [11], [24], [25]. All modern day architectures have temperature sensors built in to the hardware. Temperature of CPU, memory, storage devices and mother board etc. can be obtained using softwares like 'lm-sensors' (linux monitoring – sensors), 'lavalys' [26] and 'openhardwaremonitor' [27].

All grid resources that are involved in the execution process of jobs will periodically monitor their temperatures of i) CPU, ii) Memory, iii) Hard Disk and iv) Motherboard. If temperatures of all devices are working under normal ranges, then it will be considered as 'information' and job will continue processing with its normal checkpointing intensity. In normal checkpointing a job will send its checkpoint information after every 25% of its execution. If the temperature of the devices starts increasing or decreasing and passes out from the normal

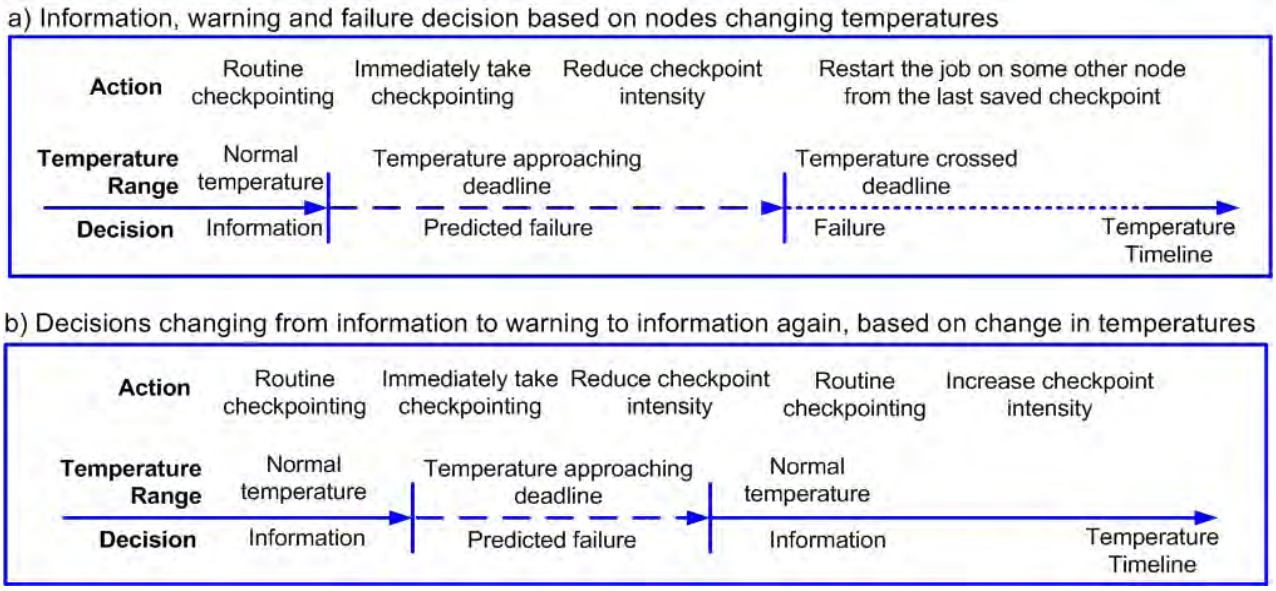

FIGURE 9. Temperature timeline.

temperature operating ranges, then this change is considered as 'prediction failure'. Failure predictor informs failure detector about failure prediction which updates the value of failure index matrix. Further actions taken for 'prediction failure' are i) reduce checkpoint intensity to 5% and ii) save checkpoint immediately.

TABLE 12. Decisions and actions on temperature ranges.

Temperature Types	Temperature Ranges	Decision	Action
Normal	Within acceptable range	Information	Take checkpoint after 25% of job execution
Approaching	Approaching Max. / Min. operating range	Warning	Take checkpoint, reduce checkpoint intensity to 5%
Crossed	Beyond Max. / Min. operating range	Failure	Restart the job to another node from last checkpoint

On reducing checkpoint intensity e.g. 50% to 25%, the number of checkpoints increases. Similarly, reducing checkpoint intensity to 5%, total number of checkpoints will be 5. From performance point of view, less number of checkpoints is recommended. Fig. 9(a) shows temperature timeline of a node where temperature based decisions are changing from 'information' to 'prediction failure' to 'hardware failure' because of the change in temperature of the device. In fig. 9(b), we can see that a node can go from 'information' to 'prediction failure' to back 'information' depending on device temperature. Decision and action taken on the basis of variation in temperature are shown in Table 12.

Failure of the device is declared when failure predictor observes device temperature beyond the maximum or minimum temperature operating range. It is the indication that device can fail any moment. Failure predictor will report this situation to failure detector as a 'hardware failure'. Working of failure predictor is presented in Algorithm 6.

J. FAILURE DETECTOR

Failure detector component receives 'prediction failure' and 'hardware' failure information from failure predictor. Upon receiving 'prediction failure' information of a resource, e.g. Ft_{rip}, it updates the failure index matrix of proactive FT orchestrator. Values of i) failure type, ii) failure intensity and iii) failure index of failure index matrix are updated upon receiving the information from failure detector. For hardware failure of resource, the value of Ft_{rih} is updated in failure index matrix of proactive FT orchestrator and values of i) failure type, ii) failure intensity and iii) failure index are also appropriately updated. Functionality of failure detector is shown in algorithm 7.

Proposed system handles three types of failures, e.g. i) hardware, ii) prediction and iii) network. Hardware and prediction failures are identified on the basis of temperatures. Failure predictor component runs on those resources that are executing jobs and measures and sends the information to failure detector. Failure detector itself detects 'network failure' that might occur due to problem in communication link or network due to which a grid resource is unable to respond. Failure detector uses 'ping liveness' messages at regular intervals to grid resources involved in execution of jobs. A sent liveness message to a grid resource if not received by the detector is considered as 'network failure'. Failure detector updates the value of Ft_{rin} in failure index matrix. Due to

Algorithm 6 Failure Predictor

Input: Temperature monitoring of grid resource entities
Output: Send 'Hardware' and 'Prediction failure' types failure info. to 'Failure Detector'
Begin
1: Monitor $temp_{entities}$ of R_i where *entities* $\in$ (CPU, Memory, Hard Disk, Motherboard)
2: **if** ($temp_{entities} \rightarrow \Delta temp_{normal}$)
3: Set decision = 'information'
4: Set action = 'routine checkpointing'
5: Checkpoint intensity = 25%
6: goto step 1
7: **end if**
8: **if** ($temp_{entities} \rightarrow \Delta temp_{lower}$ OR $temp_{entities} \rightarrow \Delta temp_{upper}$)
9: Set decision = 'failure prediction'
10: Set action = 'save checkpoint'
11: Set $time_{monitor}$ to 10 seconds
12: reduce checkpointing intensity to 5%
13: send Ft_{rip} to 'Failure Detector Module'
14: goto step 1
15: **end if**
16: **if** ($temp_{entities} \leq \Delta temp_{lower}$ OR $temp_{entities} \geq \Delta temp_{upper}$)
17: Set decision = 'hardware failure'
18: Set action = 'save checkpoint'
19: send Ft_{rih} to 'Failure Detector Module'
20: **end if**

Algorithm 7 Failure Detector

Input: Receive temperature based failure information from 'Failure Predictor'
Output: Increment failure type 'Hardware', 'Prediction' & 'Network' in 'Failure Index Matrix'
Begin
1: Receive Ft_{rip}, Ft_{rih} from 'Failure Predictor' of R_i
2: **if** (Ft_{rip} received for R_i)
3: Ft_{rip}++ for R_i in 'Failure Index Matrix';
4: **end if**
5: **if** (Ft_{rih} received for R_i)
6: Ft_{rih}++ for R_i in 'Failure Index Matrix';
7: request for new resource to 'Proactive FT Orchestrator';
8: restart the job from last saved checkpoint;
9: **end if**
10: **for** (each R_i executing jobs) **do**
11: ping all R_i for liveness after 30 second;
12: **if** (ping reply from R_i not received)
13: Ft_{rin}++ for R_i in 'Failure Index Matrix';
14: goto step 7;
15: **end if**
16: **end for**

'network failure' it is not possible to approach the resource so request for new resource is made to proactive FT orchestrator through reactive FT orchestrator. Job restarts from the last saved checkpoint on the newly provided resource. So, 'network failures' are detected through failure detector whereas temperature based 'hardware' and 'prediction failures' are identified through failure predictor.

K. INTERACTION BETWEEN ENTITIES OF GRIDSIM AND PROPOSED MODEL

In order to integrate the feature of fault tolerance in the grid, we incorporate new entities in Gridsim toolkit. The protocol responsible for interaction between Gridsim entities and entities of our proposed fault tolerant scheduling model is shown in fig. 10 and is described as follows:

1) At the start of Gridsim, all available simulated resources of grid are registered with GIS.
2) User submits the job for execution to the grid.
3) Job submitted for execution is received by 'Grid Scheduler' (Gridsim component).
4) Scheduler requests GIS to provide information about resources for the execution of the job.
5) GIS further requests proactive FT orchestrator for provisioning of information regarding resources needed for the execution of the user job.
6) In order to response the request of GIS made in 5, proactive FT orchestrator uses two of its sub-components, e.g. i) Resource Filtration and ii) Genetic algorithm based Optimal Resource Identification
   a) 'Resource Filtration' component identifies ranking on the basis of i) resource locality, ii) availability and iii) reliability according to Table 1, Table 3 and Table 6.
   b) Weightage values are placed in 'Resource Filtration Matrix' that filters out i) remotely connected resources, ii) resources having Low MTBA and iii) unreliable resources. Filtration matrix score is generated for each resource on the basis of entered values.
   c) Resources having 1 value in filtration matrix score are the filtered resources.
   d) Filtered resources are sent to GORI for selection of optimal nodes.
   e) Optimal resource pool generated by GORI is passed to GIS.
7) GIS passes optimal resource pool to 'Grid Scheduler'.
8) Scheduler selects required number of resources and forwards them to reactive FT orchestrator.
9) Reactive FT orchestrator dispatches the job to the grid resource(s) with job id.
   a) Each grid resource is equipped with failure predictor component. Resources executing user jobs keep monitoring the temperature entities of CPU, Memory, Storage devices and Motherboard through temperature sensors.

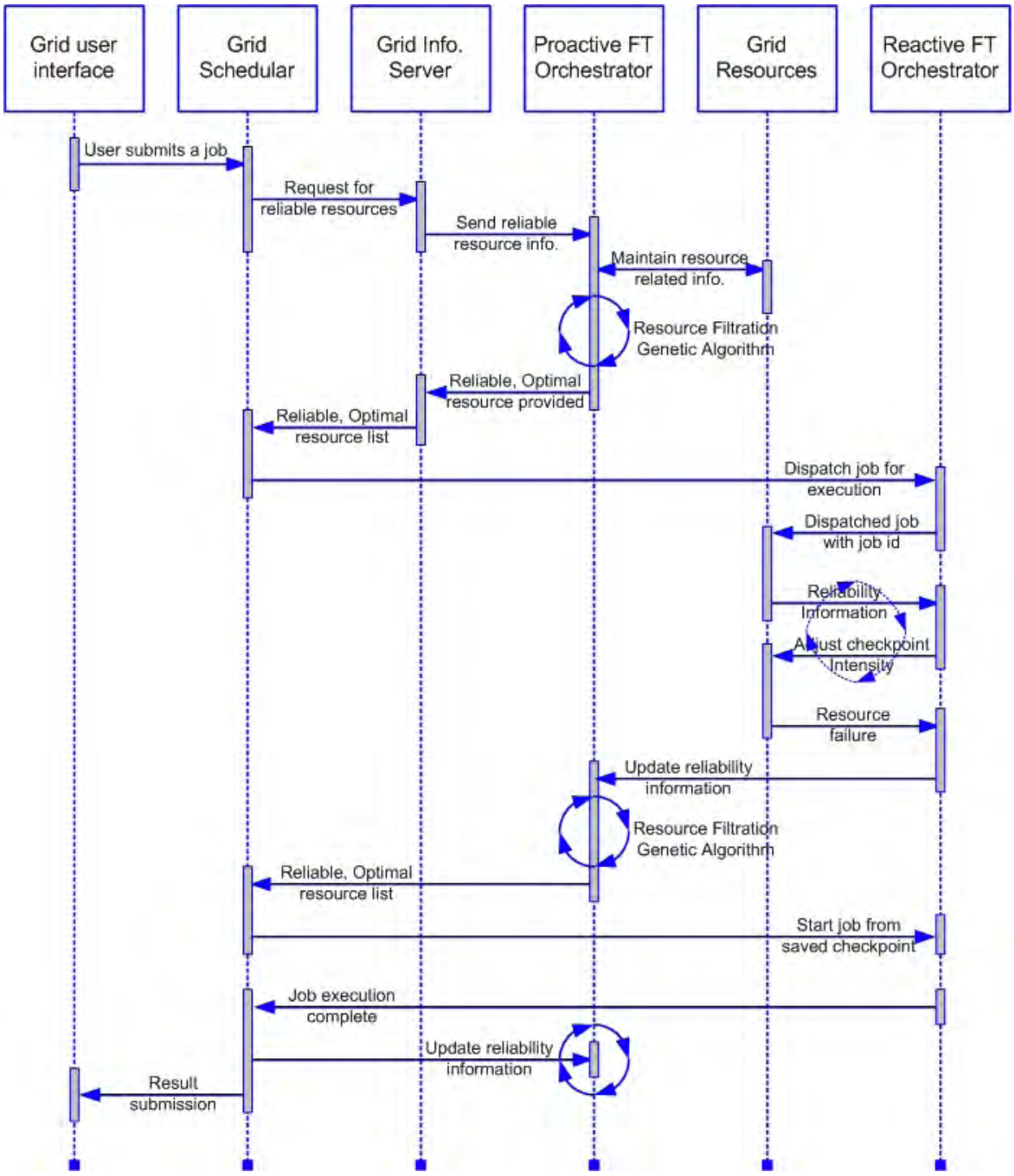

FIGURE 10. Sequence diagram for interaction between Gridsim and fault tolerant scheduling entities.

b) Normal checkpointing (after 25% execution of the job) is continued for the devices working under normal temperature operating ranges.
c) If the temperature entity of a resource increases or decreases from normal temperature operating ranges, then it is considered as 'prediction failure', and change in the behavior of temperature is notified to failure detector. Failure detector updates failure index matrix (Table 4) and reduces checkpoint intensity to 5%, e.g. from 25% to 20%.
d) If the temperature entity of a resource goes beyond the maximum or minimum temperature operating ranges, then failure detector is informed

about this change that considers it a 'hardware failure' and after saving the checkpoint, shifts the job to other available resource and updates failure index matrix (Table 4).

e) Reactive FT orchestrator contains a sub-component named failure detector for detecting network failures. Grid resources executing user jobs are periodically send 'ping liveness' messages by the failure detector. Reply not received within acceptable time is considered a network failure. Information about this failure is updated in failure index matrix (Table 4) and job is switched to other available resource from its last saved checkpoint.

10) Information about the completion of job execution is passed to 'Grid Scheduler' from reactive FT orchestrator. Failure index matrix (Table 4) is updated accordingly.

11) 'Grid Scheduler' forwards jobs completion results back to the user.

V. EXPERIMENTAL METHODOLOGY

We employ GridSim simulator-5.2 [28] to validate the performance of proposed fault tolerant scheduling scheme. Support for modeling and simulation of heterogeneous grid resources, modeling of application as well as of users is provided in the simulator. GridSim provides infrastructure support for i) creating application tasks, ii) management and mapping of tasks and resources. 'ResFailure' module of GridSim simulator was further modified using java 8 platform for induction of failures and failure types. Resource location, availability time, unavailability time, number of failures and failure types are entered in an 'XML' file and then brought into modified application through 'ArrayList' class. Resource filtration was performed according to algorithm and filtered resources were passed to genetic algorithm for optimal resource identification. Jobs were then executed on the resources identified through the proposed model. We use following performance metrics (i.e., wall clock time, throughput, waiting time, turnaround time, and number of checkpoints) to gauge the performance of the proposed system. We used number of jobs (i.e., gridlets) and fault injection percentage parameters to vary the experiment configuration. Same experimental setup was used for evaluation of all FTSS systems of Table 14.

A. RESOURCE MODELING

All grid resources having different capabilities, characteristics and configurations are modeled in an 'XML' file. Data in 'XML' file is taken from 'Failure Trace Archive (FTA)' datasets [29] and data sources available from 'World Wide Grid (WWG)' testbed [29].

B. WORKLOAD MODELING

A job entered for execution in GridSim is referred to as 'Gridlet' having length expressed in Million Instructions Per Second (MIPS), I/O operations in bytes and an originator or user ID. The workload modeling in our experiments contains a set of gridlets from 200 to 800 with a gradual increment of 200 gridlets. Locally connected resources have 100 Mbps to 1Gbps link speed and remotely connected resources have up to 10 Mbps link speed.

TABLE 13. Number of failures for failure injection percentages.

Gridlets	5%	10%	15%	20%	25%	30%
	Number of failures for varying fault injection percentages					
200	10	20	30	40	50	60
400	20	40	60	80	100	240
600	30	60	90	120	150	180
800	40	80	120	160	200	240

Number of failures represents fault injection. For a set of 200 gridlets having 5% fault injection, the dataset in 'XML' file contains 10 failures, 10% fault injection has 20 failures and so on. Fault injection percentages used in experiments gradually increases from 5% up to 30% with an increment of 5%. Table 13 represents the number of failures for various sets of gridlets. Number of instructions, e.g. size of a gridlet is fixed in all experiments for measuring the performance of different fault tolerant techniques.

TABLE 14. Performance evaluation creiteria.

#	Model ID	Authors	Technique applied	Parameters used
a.	SIS	Amoon et al.	Scheduling indicator based scheduling (SIS)	- Throughput - Turnaround Time - Waiting Time - Unavailability
b.	PRF	Latchoumy et al.	Proactive and reactive failure handling (PRF)	- Throughput - Failure probability - Resource utilization
c.	HFT	Qureshi et al.	Hybrid fault tolerance technique (HFT)	- Throughput - Turnaround Time - Waiting Time - Transmission delay
d.	MFT	Keerthika et al.	Multiconstrained Fault Tolerance (MFT)	- Makespan - Resource utilization - Processing cost - Hit count

C. PERFORMANCE EVALUATION METRICS

As mentioned earlier, to evaluate the performance of proposed model the metrics used are i) throughput, ii) waiting time iii) turnaround time and iv) number of checkpoints. We evaluate the performance of the proposed scheme compared with existing models [3], [14], [21], [23]. The metrics considered for performance evaluation in existing models are shown in the Table 14.

VI. RESULTS AND ANALYSIS

On the basis of metrics identified in para 'C', different experiments were conducted and results were compared with

the FTSS of Table 14. The list of experiments performed, their comparison with other existing FTSS, and discussion on results and comparison is mentioned below:

A. EXPERIMENT – I: WALLCLOCK TIME

In this experiment we initially entered 200 jobs on the grid and induced 5% faults in the system with three different fault tolerance considerations, e.g. i) reactive, ii) proactive and iii) hybrid (proactive and reactive) fault tolerance.

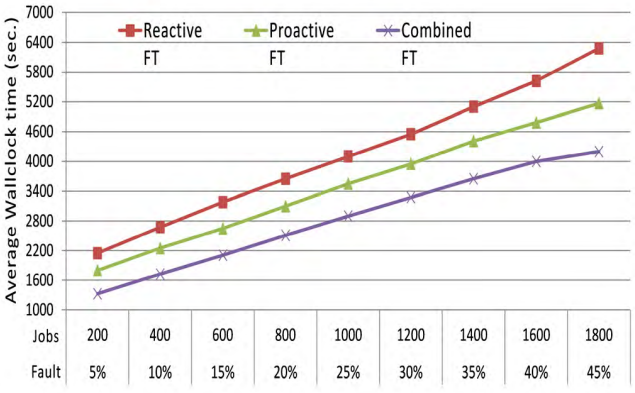

FIGURE 11. The average wallclock time, as a function of number of jobs and fault injection percentage.

Fig. 11 demonstrates the average jobs wallclock time as a function of number of jobs and fault injection percentage. The proposed hybrid scheme clearly outperforms both proactive and reactive approaches because it strives to engage locally available and reliable resources for job execution. Moreover, the average wallclock time increases with the increased number of jobs (i.e., gridlets) and fault injection percentage. The higher the gridlet count, the more time it requires to execute. With the increased fault injection percentage, more jobs need to be re-schedule. Reactive fault tolerant approach has the highest average wallclock time because it does not consider handling faults proactively.

B. EXPERIMENT – II: THROUGHPUT

Throughput is the amount of work accomplished in a given time interval. Average throughput is an important factor that helps in determining the performance of a FTSS. A robust and efficient FTSS would be that provides high throughput. If a fault tolerant scheduling system takes more time in selection of resources and takes prolonged times in saving and retrieving checkpoints etc, then throughput reduces.

Average throughput of four existing FTSS is compared with our designed system. A set of 200 gridlets were sent for execution on 'SIS', 'PRF', 'HFT' and 'MFT' FTSS. Same jobs were also executed in our proposed system (DFT). Faults with different percentages from 5 to 30% were sequentially introduced in the systems during execution and throughput obtained in the presence of those faults was recorded. Results depicted in fig. 12. and fig. 13 shows the result when average

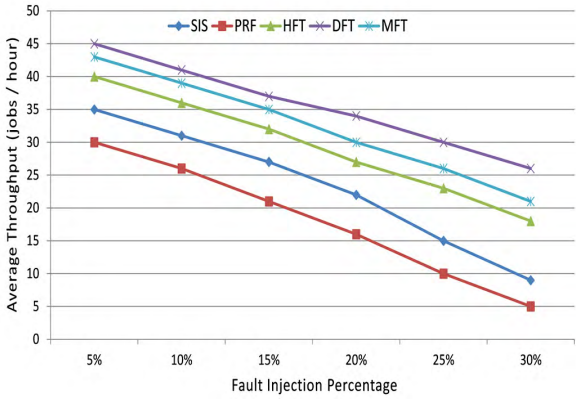

FIGURE 12. Average Throughput with 200 jobs.

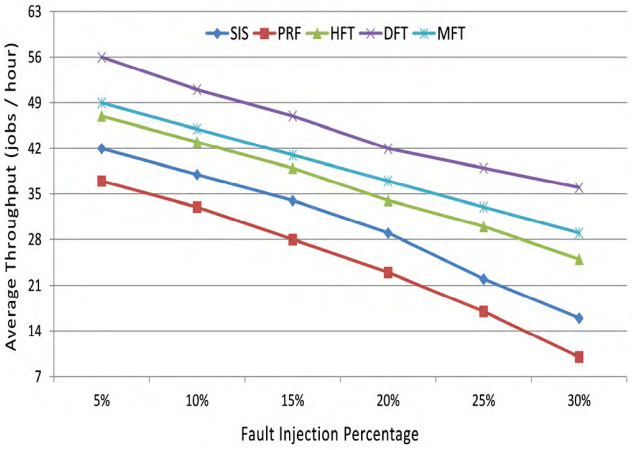

FIGURE 13. Average Throughput with 400 jobs.

throughput was calculated for 400 jobs with faults induced from 5 to 30%.

Graph of fig. 12 shows that average throughput is decreasing with increase in fault injection percentage and number of jobs entered for execution in the system. Throughput of our proposed system provides best result and reason is the selection of locally available resources that have very small or negligible communication delays, resulting in increased throughput. The model named 'PRF' has the lowest average throughput. This is due to non considerations of communication delays. When a job fails, then due to slow communication links that job has to wait for more time for completing checkpoint operations. Upon completion of checkpoint operations, the job starts from last saved checkpoint but delayed operations results in reduced average throughput. Overall average throughput difference between the compared FTSS is around 19%. Results shown in the graph of fig. 13 also shows similar type of trend observed in fig. 12.

C. EXPERIMENT – III: WAITING TIME

Waiting time of a job in the grid is the time from its submission to the time when it started execution. If grid scheduler is fast and can find resource(s) quickly, preferably optimal

resource(s) in less time, for the submitted job then waiting time of the job will be low. Grid schedulers that take more time in selection of resources are responsible for increased waiting time. A better fault tolerant scheduling system will reduce the waiting time of jobs by maintaining a list of suitable resources in advance, e.g. before the arrival of resource request from jobs. Similarly, in case of failures, the jobs will be switched in relatively lesser times to other resources, provided they also implement a good checkpointing mechanism.

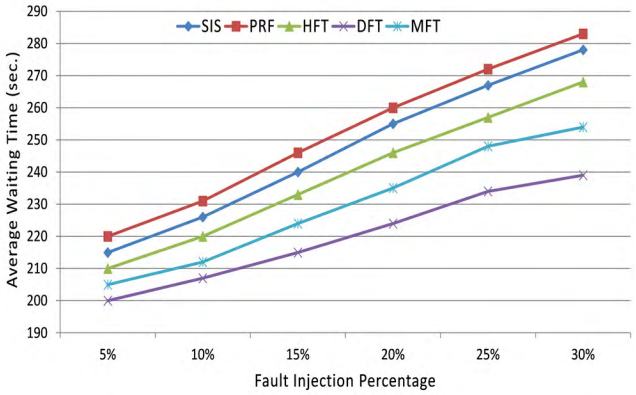

FIGURE 14. Average Waiting Time with 200 jobs.

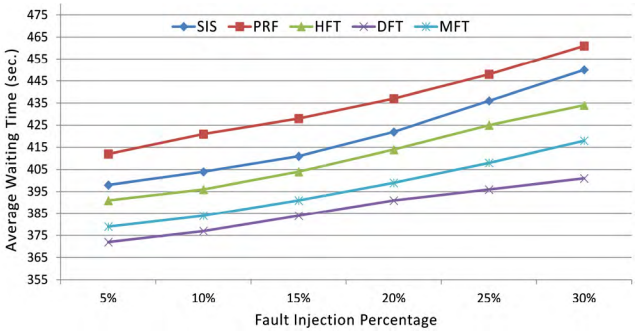

FIGURE 15. Average Waiting Time with 400 jobs.

Graphs of fig. 14 and fig. 15 show that our proposed model provides least average waiting time for all scenarios, e.g. with 200 and 400 jobs. A general observation is that waiting time or average waiting time of jobs is going to increase with the increase in number of jobs. Addition of the factor of fault injection with increasing intensities, e.g. 5% to 30% further increases waiting time. Fault tolerance schedulers, where jobs are dispatched on optimal resources and have reduced checkpoint intensities will provide better results. Our proposed model preferably selects locally connected and reliable resources and further selects optimal resources to be used in execution of jobs. This leads to less probability of network failures. Similarly, checkpoint intensity of 25% is reduced to 20% upon receiving information about 'prediction failures'. Reduction in 5% checkpoint intensity increases the number of checkpoints from 4 to 5. As resources are connected locally so it takes small amount of time in saving checkpoint information. Benefit obtained is least average waiting time compared to rest of the models.

The FTS model presented by [14] has the highest average waiting times in the fig. 14 and fig. 15. This is due to the complexity involved in resource selection. Resources are selected by generating an overall resource value through a complex system that results in delayed selection of resources. Delays in resource selection increase the waiting time and overall impact is delay in average waiting as well as in average turnaround time.

D. EXPERIMENT – IV: TURNAROUND TIME

Turnaround time is the calculation of time period from job submission to completion. Turnaround time of a resource is the sum of waiting time (WT_t), communication link time (CL_t), failure handling time (FP_t), and job execution time (JE_t). Following equation can be used for calculating the turnaround time of a job as:

$$TATJ_i = WT_t + CL_t + FP_t + JE_t; \tag{5}$$

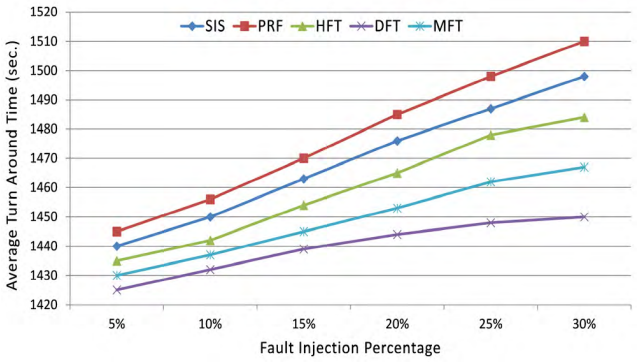

FIGURE 16. Average Turnaround Time with 200 jobs.

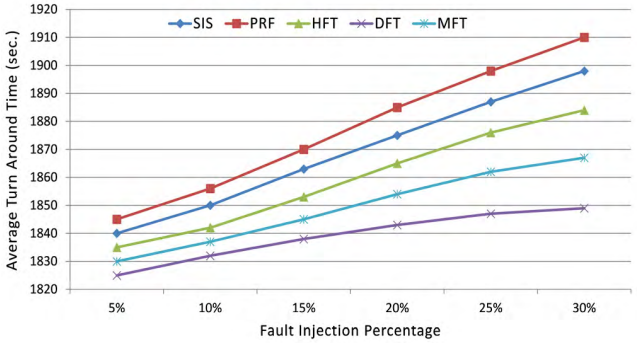

FIGURE 17. Average Turnaround Time with 400 jobs.

Turnaround time of a job will always be greater than any of the individual times mentioned in equation (5). Fig. 16 and fig. 17 show the graph of average turnaround time of the models mentioned in Table 14 along with their comparison with our proposed model. Our proposed

model outperforms in turnaround time then the rest of the models. This is due to less waiting time already discussed in graphs of sub section C. Communication time in our FTSS is negligible as our algorithm prefers locally connected resources. Our proposed model completes job execution in less time due to executing jobs on optimal resources identified through GORI. Sum of all above mentioned time produces least turnaround time compared to rest of FTSS.

FTSS [14], [21] lack proper failure detector due to which it takes more time in selection of resources that leads to increased turnaround time. Communication link is another important factor due to which the turnaround time increases in FTS models of [14], [21], and [23]. In grid, communication with remote resources is slow which increases turnaround time of the job.

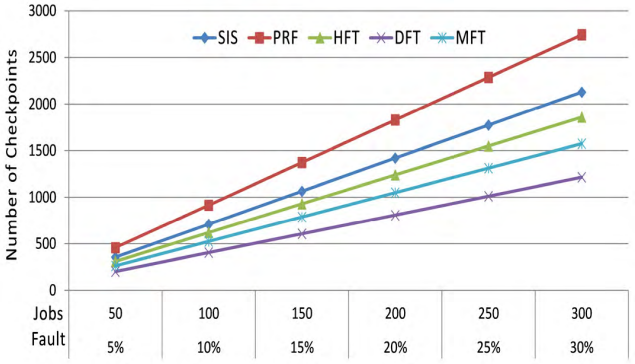

FIGURE 18. Number of checkpoints with varying number of jobs and faults.

E. EXPERIMENT – V: CHECKPOINTS

In this experiment we initially entered a set of 50 jobs with 5% faults for execution on FTSS of Table 14 and on our proposed model. The number of jobs was incremented by 50 up to 300 along with the increment in fault percentage of 5% up to 30% respectively with regular intervals depicted on horizontal axis of the graph. The number of checkpoints used by FTSS of Table 14 and our proposed model was recorded and depicted in fig. 18.

Our proposed model selects available, reliable and optimal resources for execution of jobs. Normal checkpoint intensity in our model is 25% that further reduces 5% and comes down to 20% upon identification of prediction failures. For the case of network and hardware failures, our system shifts the job on other available resource. So, the number of checkpoints required in our model are least compared to the rest of the models. The model named 'SIS' [21] used the highest number of checkpoints and our proposed model used least number of checkpoints. FTS model presented in [21] uses resource failure history and maintains a value representing number of successful and unsuccessful jobs executed on a resource and takes scheduling and selection decisions on this basis. It does not consider number of failures so at each failure it will increase number of checkpoints. Fig. 18 shows checkpoints required in each FTSS.

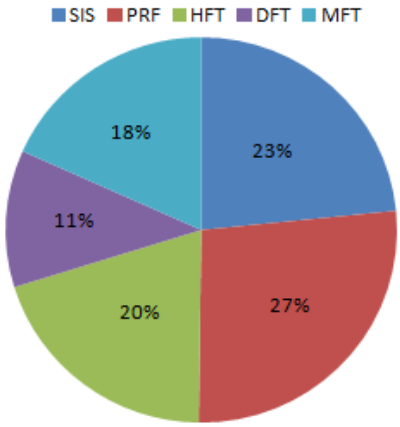

FIGURE 19. Energy consumption of different FTSS.

F. EXPERIMENT VI – ENERGY CONSUMPTION

In this experiment energy consumption of designed system and FTSS of Table 14 is compared. A compute intensive job under fixed time constraint of 3600 seconds was sent for execution on FTSS of Table 14 and our model with assumptions of no failures. All FTSS divided the job into sub jobs and executed sub jobs on various available grid resources. Number of resources selected for job execution by each FTSS, and CPU power dissipation of those resources according to their thermal design power was calculated. Energy consumption of resources under each FTSS was recorded during execution and is shown in the form of a graph in fig. 19.

It can be observed from fig. 19 that our model selected fewer resources due to which overall energy consumption in our system was less compared to other systems. Our system selected computationally faster resources that resulted timely job execution completion with less number of resources. Less number of optimal resources consumed less energy and completed execution within defined time period. Other FTSS though completed jobs within the allotted time period but due to random selection of resources with different processing capabilities utilized more resources resulting in increased usage of energy consumption. Increased energy utilization further enhances the cost of communication. Our model consumed less energy compared to all other models.

Unit wise consumption of energy, cost and time is also recorded during execution and is shown in fig. 20. All FTSS consumed 3600 units of time visible in the graph. Energy consumed based on number of resources and cost calculation in units show that our model consumed least energy and was cost effective compared to rest of the systems. Selection of computationally fast resources through genetic

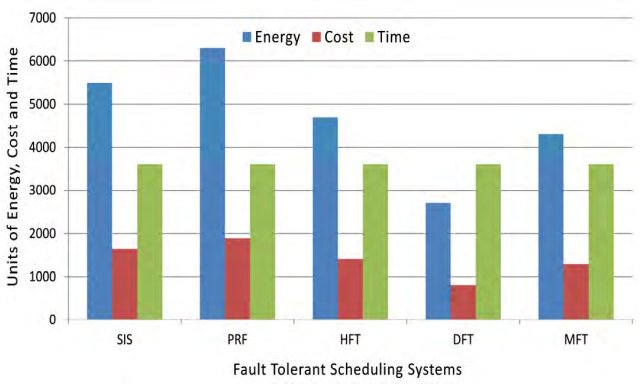

FIGURE 20. Cost, energy and time consumption.

algorithm resulted in less number of required resources for job execution. Overall cost and energy consumption was less compared to other FTSS that randomly selected more resources having less computational capabilities resulting in increased cost due to more energy requirements.

VII. CONCLUSION

In this paper, we proposed a dynamic and adaptive fault tolerant scheduling for computational grids. Our fault tolerant scheduler is a combination of proactive and reactive fault tolerant orchestrator. Proactive fault tolerant orchestrator identifies reliable resources through use of two new concepts i) location and ii) availability. Computationally optimal resources are identified through genetic algorithm. Reactive fault tolerant orchestrator dispatches resources for execution and periodically checks them for errors through monitoring temperature of the devices. Proposed model was implemented using GridSim. Results obtained through experiments and their comparison with existing models leads us to the conclusion that our proposed model is suitable for use in computational grids. Our designed model offers several benefits like increased reliability in addition to failures. Better availability of resources in a heterogeneously connected environment and provision of optimal resource selection that ensures improvement in quality of service. In future, we will extend the idea by including artificial neural network based intelligent fault detection and prediction for handling fault tolerance in grid and other HPC based systems, like cloud.

REFERENCES

[1] M. Altenbernd and D. Göddeke, "Soft fault detection and correction for multigrid," *Int. J. High Perform. Comput. Appl.*, Feb. 2017. [Online]. Available: http://dx.doi.org/10.1177/1094342016684006
[2] S. Haider and B. Nazir, *Fault Tolerance in Computational Grids: Perspectives, Challenges, and Issues*, vol. 5. London, U.K.: Springer, Nov. 2016, p. 1991.
[3] P. Keerthika and P. Suresh, "A multiconstrained grid scheduling algorithm with load balancing and fault tolerance," *Sci. World J.*, vol. 2015, May 2015, Art. no. 349576.
[4] I. Foster and C. Kesselman, *The Grid 2, Blueprint for a New Computing Infrastructure*. New York, NY, USA: Elsevier, 2003.
[5] F. U. Ambursa, R. Latip, A. Abdullah, and S. Subramaniam, "A particle swarm optimization and min–max-based workflow scheduling algorithm with QoS satisfaction for service-oriented grids," *J. Supercomput.*, pp. 1–34, Nov. 2016.
[6] H. Lee *et al.*, "A resource management and fault tolerance services in grid computing," *J. Parallel Distrib. Comput.*, vol. 65, no. 11, pp. 1305–1317, Nov. 2005.
[7] D. P. Chandrashekar, "Robust and fault-tolerant scheduling for scientific workflows in cloud computing environments," Dept. Comput. Informat. Sci., Ph.D. dissertation, Univ. Melbourne, Melbourne, VIC, Australia, 2015.
[8] A. Wadhwa and A. Bala, "Preventing faults: Fault monitoring and proactive fault tolerance in cloud computing," in *Proc. Int. Conf. ICT Sustain. Develop.*, Feb. 2016, pp. 665–673.
[9] T. Herault and Y. Robert, *Fault-Tolerance Techniques for High-Performance Computing*. New York, NY, USA: Springer, 2015.
[10] M. N. Cheraghlou *et al.*, "A survey of fault tolerance architecture in cloud computing," *J. Netw. Comput. Appl.*, vol. 61, pp. 81–92, Feb. 2016.
[11] I. P. Egwutuoha, "A proactive fault tolerance framework for high performance computing (HPC) systems in the cloud," Ph.D. dissertation, Univ. Sydney, Sydney, NSW, Australia, 2014.
[12] M. Imran, I. A. Niaz, S. Haider, N. Hussain, and M. A. Ansari, "Towards optimal fault tolerant scheduling in computational grid," in *Proc. Int. Conf. Emerg. Technol.*, Nov. 2007, pp. 154–159.
[13] F. P. Guimaraes, P. Célestin, D. M. Batista, and G. N. Rodrigues, "A framework for adaptive fault-tolerant execution of workflows in the grid: Empirical and theoretical analysis," *J. Grid Comput.*, vol. 12, no. 1, pp. 127–151, Mar. 2014.
[14] P. Latchoumy and P. S. A. Khader, "A combined approach: Proactive and reactive failure handling for efficient job execution in computational grid," in *Proc. Int. Comput. Netw. Inf.*, 2014, pp. 713–725.
[15] R. Garg and A. K. Singh, "Adaptive workflow scheduling in grid computing based on dynamic resource availability," *Eng. Sci. Technol., Int. J.*, vol. 18, no. 2, pp. 256–269, Jun. 2015.
[16] S. M. Abdulhamid, M. S. A. Latiff, S. H. H. Madni, and M. Abdullahi, "Fault tolerance aware scheduling technique for cloud computing environment using dynamic clustering algorithm," in *Proc. Neural Comput. Appl.*, Jul. 2016, pp. 1–15.
[17] M. Rebbah, Y. Slimani, A. Benyettou, and L. Brunie, "A decentralized fault tolerance model based on level of performance for grid environment," *Cluster Comput.*, vol. 19, no. 1, pp. 13–27, Mar. 2016.
[18] M. Chtepen, F. H. A. Claeys, B. Dhoedt, F. De Turck, P. A. Vanrolleghem, and P. Demeester, "Providing fault-tolerance in unreliable grid systems through adaptive checkpointing and replication," in *Proc. Int. Conf. Comput. Sci.*, 2007, pp. 454–461.
[19] X. Liu, X. Xu, X. Ren, Y. Tang, and Z. Dai, "A message logging protocol based on user level failure mitigation," in *Proc. Int. Conf. Algorithms Archit. Parallel Process.*, Dec. 2013, pp. 312–323.
[20] B. Nazir, K. Qureshi, and P. Manuel, "Replication based fault tolerant job scheduling strategy for economy driven grid," *J. Supercomput.*, vol. 62, no. 2, pp. 855–873, Nov. 2012.
[21] M. Amoon, "A fault-tolerant scheduling system for computational grids," *Comput. Elect. Eng.*, vol. 38, no. 2, pp. 399–412, Mar. 2012.
[22] B. Nazir, K. Qureshi, and P. Manuel "Adaptive checkpointing strategy to tolerate faults in economy based grid," *J. Supercomput.*, vol. 50, no. 1, pp. 1–18, Oct. 2009.
[23] K. Qureshi, F. G. K. Paul, P. Manuel, and B. Nazir, "A hybrid fault tolerance technique in grid computing system," *J. Supercomput.*, vol. 56, no. 1, pp. 106–128, Apr. 2011.
[24] K. Charoenpornwattana *et al.*, "A neural networks approach for intelligent fault prediction in HPC environments," in *Proc. High Availability Perform. Comput. Workshop (HAPCW)*, Denver, Colorado, 2008.
[25] S. Haider and N. R. Ansari, "Temperature based fault forecasting in computer clusters," in *Proc. 15th Int. Multitopic Conf. (INMIC)*, Dec. 2012, pp. 69–77.
[26] (2017). LAVALYS. *Incorporation*, accessed on Mar. 2017. [Online]. Available: http://www.lavalys.com.
[27] (2017). *Open Hardware Monitor*, accessed on Mar. 2017. [Online]. Available: http://openhardwaremonitor.org/
[28] R. Buyya and M. Murshed, "GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing," *Concurrency Comput., Pract. Exper.*, vol. 14, nos. 13–15, pp. 1175–1220, Nov./Dec. 2002.
[29] FTA. *Failure Trace Archieve, for Improving the Reliability of Distributed Systems*, accessed on Feb. 2017. [Online]. Available: http://fta.scem.uws.edu.au/

SAJJAD HAIDER received the M.S. degree in computer science in 2007. He is currently pursuing the Ph.D. degree with the Shaheed Zulfiqar Ali Bhutto Institute of Science and Technology, Islamabad, Pakistan. He is currently Faculty Member with the National University of Modern Languages (NUML), Islamabad. Before joining NUML in 2002, he was working as a Network Manager with the National Institute of Electronics, a Research and Development Organization under the Ministry of Science and Technology, Government of Pakistan. He has authored or co-authored many publications in national and international conferences and in refereed journals. His research interests include parallel and distributed computing, high performance computing, fault tolerant computing, and network security.

BABAR NAZIR is currently an Assistant Professor with the Department of Computer Science, COMSATS Institute of Information Technology, Abbottabad, Pakistan. He has authored or co-authored many prominent publications in his research fields. His current research interests include wireless sensor networks, resource management and job scheduling in cloud computing, grid computing, and cluster computing.