FTMXT: Fault-Tolerant Immediate Mode Heuristics in Computational Grid

Sanjaya Kumar Panda, Pabitra Mohan Khilar, and Durga Prasad Mohapatra

Abstract Fault tolerance plays a key role in computational grid. It enables a system to work smoothly in the presence of one or more failure components. The components are failing due to some unavoidable reasons like power failure, network failure, system failure, etc. In this chapter, we address the problem of machine failure in computational grid. The proposed system model uses the round trip time to detect the failure, and it uses the checkpointing strategy to recover from the failure. This model is applied to the traditional immediate mode heuristics such as minimum execution time (MET) and minimum completion time (MCT) (defined as MXT). The proposed Fault-Tolerant MET (FTMET) and Fault-Tolerant MCT (FTMCT) heuristics (defined as FTMXT) are simulated using MATLAB. The experimental results are discussed and compared with the traditional heuristics. The results show that the proposed approaches bypass the permanent failure and reduce the makespan.

Keywords Immediate mode • Minimum execution time • Minimum completion time • Scheduling • Fault tolerance • Grid computing

1 Background

Computational grid is widely used for high-performance computing applications [1]. It solves the large-scale complex problems by sharing and aggregation of resources [2]. The resources in grid are distributed worldwide and it is under different domains. One domain may have one or more instances. In order to

S.K. Panda (✉)
Department of CS&E, ISM Dhanbad, Dhanbad, Jharkhand, India
e-mail: sanjayauce@gmail.com

P.M. Khilar • D.P. Mohapatra
Department of CS&E, NIT Rourkela, Rourkela, OD, India
e-mail: pmkhilar@nitrkl.ac.in; durga@nitrkl.ac.in

maintain this, the Grid Referral Service (GRS) keeps track of all information about the domain and its instances [3]. It provides information to the Grid Machine/ Resource Broker (GMB/GRB). However, the broker is responsible for mapping the jobs and the available resources. It also splits the job into a number of small units called task. The scheduling model is shown in Fig. 1. The grid consumer submits the job to the broker. Then, the broker obtains the available machine list information from the GRS [4]. Finally, it maps the jobs to the resources based on the user requirements (i.e., task machine lists) and gets back the results (responses). The timeline sequence is shown in Fig. 2. In the grid, the heuristics are categorized into two types: immediate and batch mode. Immediate mode heuristic assigns a task to a resource as soon as the task arrives. Batch mode heuristic assigns a group of tasks to the resources at prescheduled times [5]. As resources are under different domains (refer Fig. 1), it may enter or leave at any point of time. Sometimes, it leads to burden on the grid [6]. Furthermore, the unpredictable resources are changing over time [7]. It may be interrupted by the domain administrator or fail due to some unavoidable circumstances. It drastically impacts on the scheduling as well as makespan.

Moreover, there are three types of failure in task scheduling. They are transient failure, intermittent failure, and permanent failure [8]. In this chapter, we have proposed a heuristic to handle the permanent failure and fail-stop model [9].

The rest of this chapter is organized as follows: Section 2 briefly discusses related research in fault tolerance and grid scheduling. Section 3 gives the preliminaries. In Sect. 4, two proposed heuristics are discussed and pseudocodes are presented. We conduct experiments and discuss results in Sect. 5. We conclude in Sect. 6.

2 Related Work

Many researchers have proposed various heuristics to reduce the makespan and increase resource utilization. But few authors address the problem of fault tolerance [10–14]. The problem of fault tolerance is of two types: resource fault (or failure) and network fault (or failure). In resource failure, the resource is not able to execute any task. But in network failure, the task is not able to arrive at the resource. Until the broker gets back the result, it may not possible to predict the failure, i.e., machine or network failure [15].

Nazir et al. [10] present the problem of fault tolerance in the form of machine failure. In this scheme, the grid information service maintains the history of fault occurrence. The value of the resource fault occurrence index is ranging from 0 to 6. If the fault index is zero, then the job is assigned to the resource. If it is in between 1 and 3, then checkpoint is appended at 50 %. So it returns the result to the broker after the 50 % execution. Similarly, if it is in between 4 and 6, then checkpoint is appended at 20 % because there is a high chance of failure.

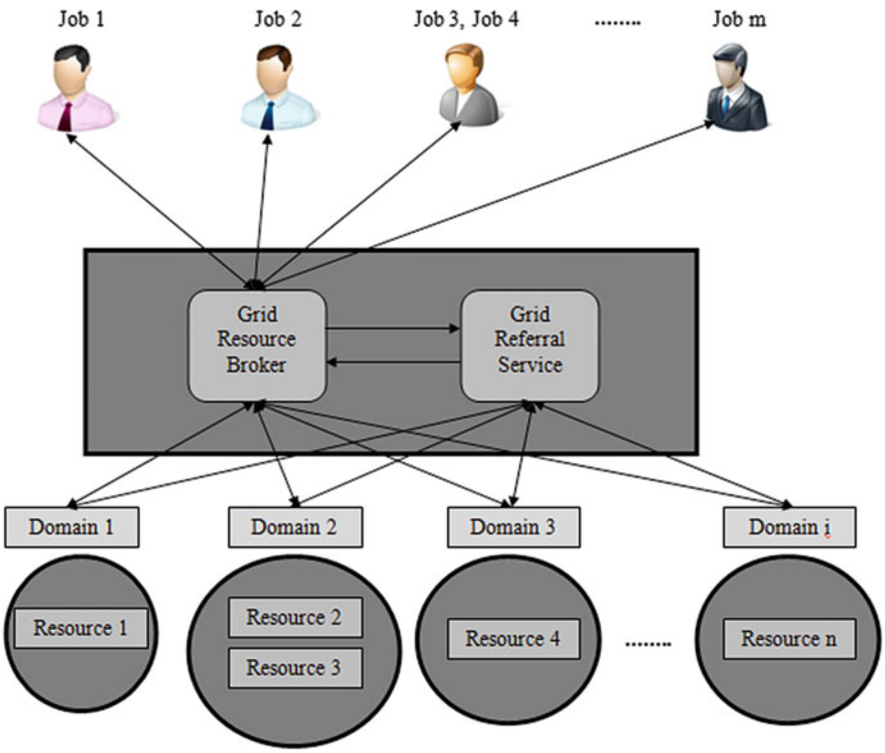

Fig. 1 A scheduling model

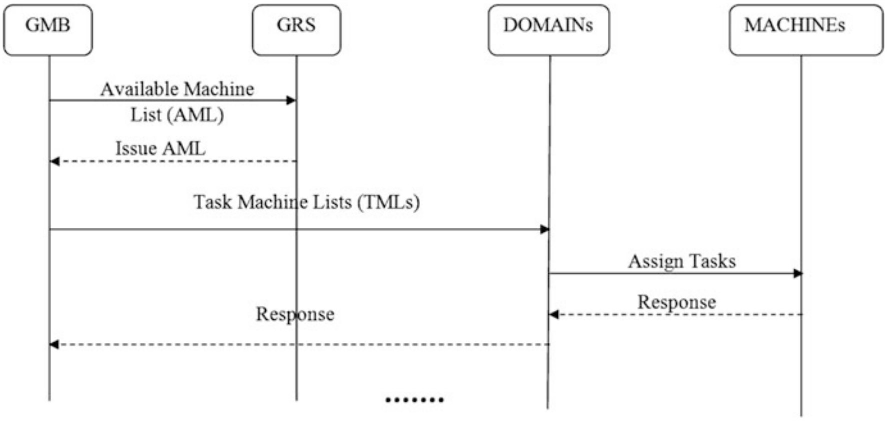

Fig. 2 Timeline sequence

Khanli et al. [12] maintain a fault occurrence history in grid information service. It uses genetic algorithm to schedule the jobs and checkpointing strategy to make scheduling more efficient. Upadhya et al. [13] propose a fault-tolerant technique based on checkpointing and passive replication. It uses genetic algorithm to perform the scheduling. To achieve reliability in a grid system, Guo et al. [11] introduce local node fault recovery mechanism. This mechanism uses ant colony optimization to solve multi-objective task scheduling. A two-phase load balancing algorithm is introduced by Nanthiya et al. [14]. First, resource selection is performed using deadline and its fault tolerance factor. Second, the load balancing algorithm is used to assign the resources.

In this chapter, we have proposed two immediate mode heuristics using round trip time and checkpointing concept. The heuristics are applied to the benchmark data set by Braun et al. [16]. To the best of our knowledge, this is the first immediate mode fault tolerance work using the benchmark data set. This chapter is an extension of our previous work [17, 18].

3 Preliminaries

3.1 Problem Statement

Assume that a grid scheduler has m different types of jobs (or tasks) $J_1, J_2, \ldots, J_m$ and n different types of resources (or machines) $R_1, R_2, \ldots, R_n$. The problem is to design an efficient heuristic to map all jobs to the available resources such that the overall processing time is minimized. In addition to this, if a resource has failed due to some unavoidable circumstances, then the proposed heuristic is able to handle it. In this chapter, we have considered the problem as the fault tolerance problem in computational grid.

3.2 Scheduling Algorithms

There are many heuristics in immediate mode heuristic. But we have listed only two heuristics: MET and MCT.

3.2.1 MET

It maps a task to a resource which takes least execution time. The main demerit of this heuristic is load imbalance because it is not considering the resource load. If the least execution time resources have failed due to some unavoidable circumstances, then this heuristic will not execute a single task. This heuristic takes O(n) time to map a task to a resource [5].

3.2.2 MCT

It maps a task to a resource which takes least completion time. The completion time is the sum of the execution time and resource ready time. This heuristic is a benchmark for immediate mode heuristic. But this heuristic is not considering the fault tolerance aspects. This heuristic takes O(n) time to map a task to a resource [5].

4 Proposed Heuristics

4.1 Description

The proposed heuristics are divided into two phases: matching and scheduling. The matching phase is similar to the traditional MET (or MCT) heuristic. The proposed fault-tolerant technique is introduced in scheduling phase.

4.2 Heuristics

The main mechanism of FTMET heuristic is defined in Algorithm 1. This heuristic maps a task to a machine which takes least execution time (lines 1 to 3). Thereafter, the broker finds the current status of the machine from GRS. Then, the GRS checks whether the machine is able to compute a task or not. It sends a message to the machine and waits for a confirmation message. If GRS will not get back the confirmation message in stipulated time, it concludes that the machine has failed or is unable to respond to the request. For that, it calculates round trip time (RTT), which is the sum of the time to send a message (to the machine) and acknowledge for it. Finally, the GRS informs the broker about the current status of the machine, and the GMB reschedules the task to the next least execution time machine. Then again, it finds the current status of the machine from GRS. If the machine works normally, then the task is assigned to the machine. Lines 4 to 12 show the scheduling process. The above process is also followed in our previous work [17, 18]. In addition, the mathematical representation is also shown over there.

Checkpointing strategy is used to recover the failure. If timeout occurs, then the GRS requests the broker to roll back to the last consistent state. Furthermore, the GRS is not sending any task to that failed (or timeout) machine.

```
Algorithm 1. FTMET heuristic
1. for task $T_i$
2.   for all machine $M_j$
3.      Find minimum $E_{i,j}$ and machine $M_j$ that holds it. //
        $E_{i,j}$ = Execution Time
4.     Set K = 1.
5.     Find the status of machine $M_j$ from GRS.
6.        if ($M_j$ == Faulty)
7.        Find (K + 1) minimum $E_{i,j}$ for task $T_i$ and machine
          $M_j$ that holds it.
8.           Go to Step 5.
9.   else Assign task $T_i$ to machine $M_j$.
10.      end if
11.    end for
12. end for
```

The main mechanism of FTMCT heuristic is defined in Algorithm 2. This heuristic maps a task to a machine which takes least completion time (lines 1 to 7). Thereafter, the broker finds the current status of the machine from GRS. If the machine fails, then the broker reschedules the task to the next least completion time machine, and it again finds the current status of the machine from GRS. Otherwise, the task is assigned to the machine. Lines 8 to 15 show the scheduling process. Like FTMET, this heuristic uses RTT and the checkpointing method in similar fashion.

```
Algorithm 2. FTMCT heuristic
1. for task $T_i$
2.   for all machine $M_j$
3.        $C_{i,j} = E_{i,j} + R_j$ // $C_{i,j}$ = Completion Time, $R_j$ =
          Ready Time
4.           end for
5. end for
6. for task $T_i$
7.   Find minimum $C_{i,j}$ and machine $M_j$ that holds it.
8.   Set K = 1.
9.   Find the status of machine $M_j$ from GRS.
10.      if ($M_j$ == Faulty)
11.          Find (K + 1) minimum $C_{i,j}$ for task $T_i$ and
             machine $M_j$ that holds it.
12.        Go to Step 9.
13.      else Assign task $T_i$ to machine $M_j$.
14.    end if
15. end for
```

Table 1 Expected execution time of four tasks on three machines

Task	M_1	M_2	M_3
T_1	120	75	32
T_2	40	110	93
T_3	71	24	49
T_4	23	34	47

5 Experimental Study

In this section, we present the results in terms of two performance measures makespan and machine utilization. Makespan is the overall completion time taken to assign all tasks to the machines. Machine utilization is the percentage of time the machine is busy. The results are carried out using MATLAB.

Let us consider a 4×3 matrix shown in Table 1. We assumed that tasks arrive in the following order: T_1, T_2, T_3, and T_4. In MET, task T_1 has least execution time on machine M_3. So it is assigned to machine M_3. Like task T_1, the least execution time for task T_2, task T_3, and task T_4 is machine M_1, machine M_2, and machine M_1, respectively. So the tasks are assigned to the machine accordingly. The overall makespan is 63.

Consider a scenario in which the machine M_1 has failed due to some unavoidable circumstances. So task T_2 and task T_4 are not executed successfully. In the proposed FTMET, if the machine M_1 has failed due to some unavoidable circumstances, then task T_2 and task T_4 are assigned to the second least execution time machine, i.e., machine M_3 and machine M_2, respectively. The overall makespan is 125. The major advantage is to keep the task against faults.

In MCT, task T_1 has least completion time on machine M_3. So it is assigned to machine M_3. However, for task T_2, the least completion time machine is M_1. So task T_2 is assigned to machine M_1. Similarly, task T_3 and task T_4 are assigned to machine M_2. The overall makespan is 58. Note that MET/MCT scheduling assumes that none of the machines have failed in the middle of the execution.

Consider a scenario in which the machine M_1 has failed due to some unavoidable circumstances. So the task is not executed successfully. In the proposed FTMCT, if the machine M_1 has failed due to some unavoidable circumstances, then task T_2 is assigned to the second least completion time machine, i.e., machine M_2. The overall makespan is 128.

We have considered Braun et al. [16] data sets (or instances) to evaluate the proposed heuristics. The instances are classified into 12 different types of matrices. The general form of the instance is u_t_mmnn.o. Here, u indicates the uniform distribution; t indicates the nature of the matrix: consistent (c), inconsistent (i), and semi-consistent (s); mm indicates the task heterogeneity; and nn indicates the machine heterogeneity. The value of mm or nn is either hi or lo.

In this chapter, we have taken three different sizes of the data sets: 512×16, 1024×32, and 2048×64. In 512×16 instances, we have assumed that eight numbers of machines have failed during simulation. The machine IDs are M_{10}, M_3, M_4, M_{15}, M_1, M_8, M_{16}, and M_6. The machines are failed after the execution of the task ID T_{165}, T_{176}, T_{182}, T_{188}, T_{234}, T_{314}, T_{338}, and T_{370}, respectively.

Table 2 Numerical results of makespan value for 512 × 16, 1024 × 32, and 2048 × 64 instances, respectively

Inst ance	MET (512×16)	MET (1024×32)	MET (2048×64)	FTMET (512×16)	FTMET (1024×32)	FTMET (2048×64)	MCT (512×16)	MCT (1024×32)	MCT (2048×64)	FTMCT (512×16)	FTMCT (1024×32)	FTMCT (2048×64)
u_c_hihi	2.2159E+07	3.8431E+07	1.6736E+07	5.3052E+07	7.1204E+07	3.5770E+07	1.1423E+07	3.1749E+07	2.7362E+07	2.1794E+07	9.0368E+07	7.2217E+07
u_c_hilo	5.3951E+05	3.6308E+06	1.6641E+06	8.7844E+05	7.3878E+06	3.5821E+06	1.8589E+05	3.1614E+06	2.6695E+06	3.1088E+05	9.3558E+06	7.1534E+06
u_c_lohi	6.6846E+05	3.2742E+03	1.7626E+03	1.6256E+06	6.8098E+03	3.7223E+03	3.7830E+05	2.8765E+03	2.7639E+03	7.3675E+05	9.1600E+03	7.3549E+03
u_c_lolo	1.8065E+04	3.7785E+02	1.7265E+02	2.8962E+04	7.8480E+02	3.7123E+02	6.3601E+03	3.2576E+02	2.7196E+02	1.0175E+04	9.2257E+02	7.2791E+02
u_i_hihi	3.7073E+06	6.7612E+06	4.1277E+06	8.5274E+06	1.3968E+07	5.2893E+06	4.4136E+06	7.4194E+06	3.6175E+06	9.2631E+06	1.5616E+07	5.6261E+06
u_i_hilo	9.4796E+04	6.7070E+05	4.6574E+05	1.6403E+05	1.4525E+06	5.7573E+05	9.4856E+04	6.7008E+05	4.0982E+05	1.6698E+05	1.5199E+06	5.7434E+05
u_i_lohi	1.4232E+05	8.5439E+02	4.2063E+02	2.7597E+05	1.6435E+03	5.2275E+02	1.4382E+05	7.5134E+02	3.8518E+02	3.0498E+05	1.5039E+03	5.6427E+02
u_i_lolo	3.3993E+03	9.1120E+01	3.4820E+01	5.6228E+03	1.4735E+02	6.1960E+01	3.1374E+03	6.9460E+01	4.0810E+01	5.2437E+03	1.5151E+02	5.7580E+01
u_s_hihi	1.1077E+07	2.4737E+07	9.8003E+06	3.0105E+07	4.0502E+07	2.2545E+07	6.4227E+06	1.7347E+07	1.5599E+07	1.5219E+07	5.5970E+07	4.0392E+07
u_s_hilo	2.7135E+05	2.2116E+06	8.2527E+05	3.1881E+05	3.7105E+06	1.7650E+06	1.1837E+05	1.7473E+06	1.3726E+06	2.3047E+05	5.4920E+06	3.7840E+06
u_s_lohi	3.0255E+05	2.1260E+03	8.7390E+02	8.6459E+05	3.6559E+03	1.7365E+03	1.8409E+05	1.6444E+03	1.3767E+03	4.4428E+05	5.4971E+03	3.9021E+03
u_s_lolo	8.6922E+03	1.7873E+02	8.5660E+01	1.2084E+04	4.0626E+02	2.1785E+02	4.4361E+03	1.8050E+02	1.4440E+02	8.1171E+03	5.7442E+02	4.0979E+02

Table 3 Numerical results of machine utilization value for 512 × 16, 1024 × 32, and 2048 × 64 instances, respectively

Instance	MET (512×16)	MET (1024×32)	MET (2048×64)	FTMET (512×16)	FTMET (1024×32)	FTMET (2048×64)	MCT (512×16)	MCT (1024×32)	MCT (2048×64)	FTMCT (512×16)	FTMCT (1024×32)	FTMCT (2048×64)
u_c_hihi	1	1	1	0.7088	0.6000	0.5709	0.7020	0.6475	0.6167	0.6424	0.5794	0.5957
u_c_hilo	1	1	1	0.8071	0.5580	0.5523	0.7090	0.6495	0.6168	0.6489	0.5668	0.5871
u_c_lohi	1	1	1	0.7056	0.5729	0.5494	0.7054	0.6480	0.6111	0.6320	0.5714	0.5826
u_c_lolo	1	1	1	0.8119	0.5558	0.5600	0.6980	0.6448	0.6196	0.6568	0.5758	0.5799
u_i_hihi	0.5741	0.4994	0.3781	0.4901	0.4437	0.4046	0.7077	0.6176	0.5816	0.6285	0.5609	0.5313
u_i_hilo	0.5711	0.4495	0.3522	0.5080	0.4117	0.3901	0.7016	0.6428	0.5679	0.6339	0.5570	0.5662
u_i_lohi	0.5002	0.3730	0.3855	0.5091	0.3956	0.4418	0.6961	0.6058	0.5897	0.6256	0.5855	0.5793
u_i_lolo	0.5587	0.3546	0.4494	0.4949	0.4196	0.3627	0.7202	0.6419	0.5687	0.6471	0.5785	0.5607
u_s_hihi	0.2420	0.0928	0.0953	0.3032	0.1337	0.0801	0.7119	0.6704	0.5808	0.6210	0.5671	0.5746
u_s_hilo	0.2660	0.1116	0.1208	0.4960	0.1447	0.0935	0.7339	0.6471	0.6029	0.6264	0.5591	0.5848
u_s_lohi	0.2863	0.1137	0.1094	0.3105	0.1361	0.0918	0.7103	0.6696	0.6145	0.6060	0.5439	0.5607
u_s_lolo	0.3109	0.1184	0.1083	0.4637	0.1226	0.0809	0.7093	0.6317	0.6087	0.6227	0.5527	0.5610

The numerical results of makespan value for 512×16, 1024×32, and 2048×64 instances are shown in Table 2. Similarly, the numerical results of machine utilization value for the above instances are shown in Table 3 respectively. The RTT (or timeout) and checkpointing time overhead are negligible for simplicity. In Tables 2 and 3, MET and MCT heuristics are very less makespan because very few number of tasks are executed (due to machine failure). For example, let us consider the 512×16 instances. In MET, the total number of tasks executed in u_c_hihi instances is 512–279 (total number of failed task) = 233. The makespan is 2.2159E + 07. But the proposed FTMET executes 512 tasks in the presence of failure. Hence, the makespan is 5.3052E + 07. So FTMET gives better performance in the presence of failure.

6 Conclusion

In this chapter, we have introduced two new heuristics: FTMET and FTMCT. These heuristics are an extension of the existing MET and MCT heuristic. The proposed heuristics address the problem of fault tolerance and its solution. Heuristics are evaluated using benchmark instances. The results show that the proposed heuristics give better task scheduling and minimize the makespan.

References

1. Medeiros, R., Cirne, W., Brasileiro, F., Sauve, J.: Faults in Grids: why are they so bad and what can be done about it. In: Proceedings of the Fourth International Workshop on Grid Computing. (2003)
2. Murshed, M., Buyya, R., Abramson, D.: GridSim: A Toolkit for the Modeling and Simulation of Global Grids, pp. 1–15. Monash University Journal. (2001)
3. Vasques, J., Veiga, L.: A decentralized utility-based grid scheduling algorithm. In: 28th Annual ACM Symposium on Applied Computing, pp. 619–624. (2013)
4. Li, M., Xiong, N., Yang, B., Li, Z., Park, J.H., Lee, C.: Posted price model based on GRS and its optimization for improving grid resource sharing efficiency. Telecommun. Syst. **55**(1), 71–79 (2014)
5. Maheswaran, M., Ali, S., Siegel, H.J., Hensgen, D., Freund, R.F.: Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. J. Parallel Distrib. Comput. **59**, 107–131 (1999)
6. Sadashiv, N., Kumar, S.M.D.: Cluster, grid and Cloud computing: a detailed comparison. In: IEEE 6th International Conference on Computer Science and Education, Singapore, pp. 477–482. (2011)
7. Ergu, D., Kou, G., Peng, Y., Shi, Y., Shi, Y.: The analytic hierarchy process: task scheduling and resource allocation in cloud computing environment. J. Supercomput. **64**, 835–848 (2013). Springer
8. Mushtaq, H., Al-Ars, Z., Bertels, K.: Survey of fault tolerance techniques for shared memory multicore/multiprocessor systems. In: IEEE 6th International Design and Test Workshop, Beirut, Lebanon, pp. 12–17. (2011)

9. Treaster, M.: A Survey of Fault-Tolerance and Fault-Recovery Techniques in Parallel Systems. National Center for Supercomputing Applications. University of Illinois. (2005)
10. Nazir, B., Khan, T.: Fault tolerant job scheduling in computational grid. In: IEEE 2nd International Conference on Emerging Technologies, Peshawar, Pakistan, pp. 708–713. (2006)
11. Guo, S., Huang, H., Wang, Z., Xie, M.: Grid service reliability modeling and optimal task scheduling considering fault recovery. IEEE Trans. Reliab. **60**, 263–274 (2011)
12. Khanli, L.M., Far, M.E., Rahmani, A.M.: RFOH: a new fault tolerant job scheduler in grid computing. In: IEEE 2nd International Conference on Computer Engineering and Applications, Bali, Indonesia, pp. 422–425. (2010)
13. Upadhyay, N., Misra, M.: Incorporating fault tolerance in GA-based scheduling in grid environment. In: IEEE World Congress Information and Communication Technologies, Mumbai, India, pp. 772–777. (2011)
14. Nanthiya, D., Keerthika, P.: Load balancing GridSim architecture with fault tolerance. In: International Conference on Information Communication and Embedded Systems, Chennai, India, pp. 425–428. (2013)
15. Duarte, E.P., Weber, A., Fonseca, K.V.O.: Distributed diagnosis of dynamic events in partitionable arbitrary topology networks. IEEE Trans. Parallel Distrib. Syst. **23**, 1415–1426 (2012)
16. Braun, T.D., Siegel, H.J., Beck, N., Boloni, L.L., Maheswaran, M., Reuther, A.I., Robertson, J. P., Theys, M.D., Yao, B.: A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. J. Parallel Distrib. Comput. **61**, 810–837 (2001)
17. Panda, S.K., Khilar, P.M., Mohapatra, D.P.: FTM^2: fault tolerant batch mode heuristics in computational grid. In: 10th International Conference on Distributed Computing and Internet Technology. Lecture Notes in Computer Science, vol. 8337, pp. 98–104. (2013)
18. Panda, S.K.: Efficient scheduling heuristics for independent tasks in computational grids. M. Tech. thesis, National Institute of Technology Rourkela (2013)