The Response Rank based Fault-tolerant Task Scheduling for Cloud System

Sheikh Umar Mushtaq*, Sophiya Sheikh, and Ajay Nain

Lovely Professional University Jalandhar-Delhi, G.T. Road, Phagwara, Punjab, 144411, INDIA
shiekhumar12@gmail.com

Abstract. A cloud computing platform has higher failure rates because of its highly dynamic nature and running of concurrent applications. However, the outcomes of running concurrent applications won't be accurate without VM synchronization. The issue of coordination between many virtual machines (VMs) is their synchronization in working is rarely considered by existing solutions. Moreover, fault tolerance constitutes one of the most crucial components for cloud computing architecture to ensure high reliability. In this research, Reserved Fault Tolerance and Ranked Task Scheduling (RFRTS) is proposed. Initially, the proposed ranked based scheduling approach is used for task allocation, and later reservation-based reactive fault tolerance method is suggested for a cloud system. To achieve the highest level of cloud computing infrastructure reliability, the suggested technique considers CPU faults and the VM reservation will ensure the assignment of alternative VM to the affected task. The proposed fault-tolerant approach has been compared with three existing reliable fault-tolerant approaches namely multi-objective scheduling algorithm with Fuzzy Resource utilization (FR-MOS), Cost-effective Workflow Scheduling Algorithm (CWS), and Fault-tolerant Cost-effective Workflow Scheduling Algorithm (FCWS) based on reliability. The outcomes unequivocally show that our suggested RFRTS algorithm surpasses the current FR-MOS, CWS, and FCWS considering the reliability in all the states.

Keywords: Fault, Scheduling, Response, Reliability, Rank

1 Introduction

Cloud computing have become a well-liked and frequently utilised technology in recent years, offering computing amenities to end users as well as businesses [1][2][3]. These computing models' primary benefits are virtualization, quick flexibility, pay-per-use, immediate access, and furthermore. Because of these features, cloud computing platforms are well suited to managing diverse needs and computationally intensive scientific process applications [1]–[4].

Distributed collaborative computing, however, necessitates the use of several cloud computing services due to the variety and scope of the needs for these scientific applications. This cloud computing platform is referred to as a "multi-cloud system," where

A. Putro Suryotomo and H. Cahya Rustamaji (eds.), *Proceedings of the 2023 1st International Conference on Advanced Informatics and Intelligent Information Systems (ICAI3S 2023)*,
Advances in Intelligent Systems Research 181,
https://doi.org/10.2991/978-94-6463-366-5_5

distinct commercial cloud Infrastructure as a Service (IaaS), such as the IBM Cloud and AWS S3, is managed by a third party in several countries [5], [6], and [7]. Each IaaS provider may provide a collection of VMs having various configurations and computational power [8]. By making VMs available with their own billing systems, the multi-cloud systems will share the resources of cloud providers [7]. For instance, the cloud service provider Amazon EC2 has a coarse-grained invoicing system that treats each half hour as a full hour [9]. Microsoft Azure, a cloud service provider, charges customers per minute [10]. As a result, for multi-cloud systems with different billing processes, determining how to acquire the best reliability has emerged as a significant difficulty.

Contrarily, multi-cloud computing systems are made up of a collection of diverse virtual resources connected by both an external commercial network and an internal high-speed communication network. These virtual computers contain their own software, hardware, and administration systems and are owned by several organisations. However, in these systems, cloud resources are vulnerable to a variety of problems, including network failure, hardware malfunction, delay failure, resource lacking failure, and so forth [11]. As a result, multi-cloud systems' dependability always decreases.

Numerous methods, such as checkpoint/recovery, failure prevention, fault projections, fault tolerance, and task scheduling, have been researched [12]–[13] in order to increase the reliability of scientific application execution. Among these technologies, coordinated scheduling and fault tolerance are likely the most alluring methods to increase the dependability of workflow execution. Fault-tolerant systems use backups to try to endure any cloud computing VM failure [13], [14]. The most popular method is replication, which offers a main VM and a number of redundant VMs from several cloud providers for tasks. As a consequence, the results of the backup jobs can be promptly available if the main task's execution fails due to VM failure. Regardless of the reliability of the main VM, the replication-based v-fault tolerance approaches generally need v-backup VMs [13]. However, In this paper, a method known as VM reservation has been used where the alternative VM is reserved for the task based on the pre-estimated reservation window. This reservation window makes the approach effective for reliability. Besides, tasks are initially allocated based on the proposed Response Ratio. The rest of the paper is organised as Section 2 presents the existing literature related to the work. The proposed model with problem formulation and pseudocode of the allocation and fault tolerance is presented in Section 3. Section 4 discusses the outcomes of the model concerning reliability and Section 5 concludes the paper with some future directions.

2 Related Work

Users of the cloud frequently grapple with the issue of fault tolerance. In accordance with both the non-stabilization of links to networks and the virtualization theory, all of these problems, along with others, have a negative impact on the performance, efficiency, and availability of cloud computing. Because of this, fault tolerance in cloud computing has been the subject of much research and investigation. Besides,

[15] presented a checkpoint system for virtual machines. The suggested VM-Checkpoint approach makes use of a check-pointing framework for executing the VM application and recovers an identical snapshot of the VM at a certain point in time using in-memory incremental checkpoints. The primary goals of the proposed approach are conducting in-place recovery and storing the checkpoints in memory. In [16], the system maintains the reliability of the working nodes in real-time systems. The virtual machine's dependability varies with each computing cycle and is correlated with the accuracy of findings to ensure the adaptability of the system. To tolerate failures in distributed systems, [17,18] presented a replication-based Low Latency Fault Tolerance System (LLFT). It protects the program from multiple faults and copies the application using several replication techniques. Another, Fault Tolerance Management (FTM) approach uses checkpoint and replication strategies to offer reactive fault tolerance, which is presented in [19]. Furthermore, [20] utilises the Nave Bayes predictive classifier to implement proactive fault tolerance. With the Nave Bayes classifier, it forecasts the nodes that are most likely to encounter a defect, and then it employs fault tolerance strategies to increase the system's dependability. In this chain, [21] provided an integrated fault tolerance technique that incorporates the benefits of replication strategy and checkpoint that are targeted at virtual machine failure. Additionally, Ranking and fault handling are the two operational phases of this architecture. The ranking-based framework and proactive FT-Cloud model that automatically finds defects were proposed in [22]. To increase the security of scientific process execution, an attack-defense game model based on Nash Equilibrium and dynamic Heterogeneous Earliest Finish Time (HEFT) has been developed [23]. To address the issue of system resource management, several workflow scheduling techniques that consider system dependability have been developed [24-26]. [24] introduced an RDLS method that included reliability expenses into the dynamic level of the DAG model. [25] suggested a reliability-conscious scheduling technique with duplication for cluster systems.

An ant colony system-based workflow scheduling method (CWS) was suggested by [26] in an effort to increase the dependability of applications executed under time and financial limitations. The publication [27] that suggested a multi-objective scheduling algorithm with fuzzy resource utilisation (FR-MOS) based on a particle swarm optimisation approach is the one that is most similar to our study. The objectives are to reduce costs, shorten lead times, and balance resource usage. However, the fuzzy logic approach and resource use limit the FR-MOS algorithm's capacity to optimise system reliability. There is therefore plenty of potential to raise the reliability of application execution. Similarly, a fault-tolerant cost-efficient workflow scheduling algorithm (FCWS) is presented in [28] to reduce application execution cost, time, and execution reliability by considering multi-cloud execution reliability and cloud provider billing mechanisms.

After studying and analysing the existing reliable models in the literature, it was found that there is enough room for working on the reliability of the system by guaranteeing the complete execution of every task. In this study, we introduced Response Ratio based allocation and put forth a reservation-based fault tolerance mechanism that calculates the task-specific reservation window and offers a suitable replacement VM for tasks that cannot be completed because of faults in corresponding VMs. Table 1

indicates the brief comparative analysis between the top-cited fault tolerance approaches in the literature and the proposed approach.

Table 1. Previous fault tolerance models comparison.

FT model	Resource awareness	Policies	Reliability	Implemented environment
Fault tolerance in the cloud using reactive and proactive techniques [13]	No	Reactive	N/A	Cloud simulator
AFTRC [14]	No	Proactive/reactive	Yes	Amazon EC2
LLFT [15]	No	Reactive	N/A	Cloud
FTM [17]	Yes	Reactive	N/A	Cloud simulator
An approach for fault tolerance in cloud computing using machine learning technique [20]	No	Proactive	Yes	Cloud simulator
FT-Cloud [21]	Yes	Reactive	Yes	Amazon EC2
Hybrid FT [25]	No	Reactive	Yes	Cloud simulator
Proposed RFRTS	Yes	Reactive	Yes	Self-simulation in Python

3 Proposed Work

This section presents an effective allocation to map the incoming tasks and available VMs. After the mapping of task and VM sets, we propose a framework that would estimate the reservation window based on the size of the task and the VM's capacities. This will move the affected tasks from unstable VMs to reserved and trustworthy ones based on the reservation window. The suggested model's primary goal is to increase system reliability through assured task execution using advance resource reservations.

3.1 Problem Formulation

The VMs are assigned in set V={v1, v2, v3… vk}. while the tasks being taken as set T= {t1, t2, t3… tm}. Every VM has VM capacity, i.e., (C(vk)), and every incoming task is having task length, i.e., (L(tm)). Moreover, every task (tm) has its task id (t_id) which is assigned to the task on FCFS (First Come First Serve) basis. Initially, the incoming tasks have been ranked based on the parameters of task, t_id, and L(tm). The response rank value (r) is calculated for every incoming task. Further, the ranked tasks

are taken as separate task set, i.e., Tr. The tasks are distributed in order of rank value to the corresponding VMs. The r value for the task is calculated by adding the t_id and processing time of the task and dividing the obtained value by the processing time of the task. The ranked allocation considers both the wait time and processing time of the task. It also minimizes the wait time for large tasks and simultaneously encourages the small tasks to get the higher rank thereby it gives an optimized response time than that of Shortest Job First allocation. Besides, the model also includes fault tolerance by utilizing resource reservation techniques, where the VM is reserved for the task for a specific pre-estimated window known as a reservation window (R).

Table 2. Notions used.

T	Task set	$PT(t_m)$	Processing time of t_m
V	VM set	T_r	Ranked task set
r	Response Rank Value	t_id	Task id
$CT(t_m,v_k)$	Completion Time of t_m on v_k	$L(t_m)$	Length of t_m
$RT(v_k)$	Ready Time of v_k	$R(t_m,v_k)$	Reservation window of t_m on v_k
$ST(t_m,v_k)$	Start Time of t_m on v_k	T_p	Prematurely Terminated Task set

Further, the various assumptions of the proposed allocation strategy are:

- The lower task heterogeneity metric is used by the model.
- Task sizes range from one to one hundred million instructions.
- The benchmark for low machine heterogeneity is used by the model.
- The speed of the machine varies between one to ten Million Instructions Per Second (MIPS).

The task allocation problem is mathematically represented as a mapping of each incoming task to the VM as shown in eq. (1). The bipartite graph between the task set and the VM set may be used to describe the suggested ranking approach. The set T_r contains the ranked tasks in descending order of r-value rather than the t_id, hence the tasks in T_r may be arranged randomly concerning to t_id as indicated in Fig. 1.

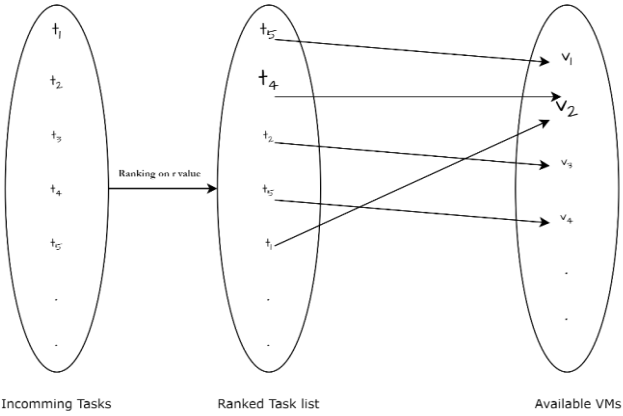

Fig. 1. Ranked task mapping.

$$\mathcal{M} : T \rightarrow V \tag{1}$$

In the event of any fault happening to any of the VMs under processing, it may have an impact on the overall processing of the tasks. The tasks might be terminated prior to completion as a result of this issue; therefore, we reserve the VMs. In this method, the execution time window for each task on an appropriate VM, known as the R window, is determined and reserved beforehand. By offering such an appropriate alternative VM, the system will continue the completion of the affected task even in case of failure. The R window for every task must be calculated using two major times, i.e., start time (ST) and completion time (CT). The ST is the point at which a certain task on the designated VM starts its execution. Besides, each VM has its load history of processing termed as Ready Time of v_k ($RT(v_k)$) which represents the point when the specific VM is set to process the new task. Once any t_m will start running on any v_k, the $RT(v_k)$ will become the $ST(t_m, v_k)$ as shown in eq. (2)

$$ST(t_m, v_k) = RT(v_k) \tag{2}$$

The CT is the time when the task on the allotted VM has completed its execution. To calculate the $CT(t_m, v_k)$, the t_m's processing time on v_k ($PT(t_m, v_k)$) is added to its $ST(t_m, v_k)$ as in eq. (3)

$$CT(t_m, v_k) = ST(t_m, v_k) + PT(t_m, v_k) \tag{3}$$

Where, $PT(t_m, v_k)$ is calculated by dividing the capacity of v_k with the length of t_m as shown in eq. (4)

$$PT(t_m, v_k) = \frac{C(v_k)}{L(t_m)} \tag{4}$$

Now, suppose "q" VMs are malfunctioning which results in premature termination of "p" associated tasks. These 'q' virtual machines are handled as a separate set of VMs in V_m as shown:

$$V_m = \{v_m : v_m \in V \text{ and } n(V_m) = q \ \& \ q < \frac{n(V)}{2}\} \quad (5)$$

Similarly, the "p" prematurely terminated associated task is taken as a separate set of tasks in T_p as shown:

$$T_p = \{t_p : t_p \in T \text{ and } n(T_p) = p \ \& \ p <= n(T)\} \quad (6)$$

Without any fault-tolerant approach, the whole task set (T_p) will not be executed. However, if T_p will be migrated to some other VMs, the execution will be continued. For enabling reservation, the R window is calculated by taking the difference of $CT(t_m, v_k)$ and $ST(t_m, v_k)$ as shown in eq. (5)

$$R(t_m, v_k) = CT(t_m, v_k) - ST(t_m, v_k) \quad (7)$$

Now, in order to finish the execution of t_ps, they must be reassigned to a different VM, which is performed via the reservation approach. For the estimated R window, all tasks in T_p are transferred from V_m to a different virtual machine in V. The reassignment of T_p is shown mathematically as:

$$T_p \rightarrow v_k \mid v_k \in V \ \&\& \ v_k \notin V_m \quad (8)$$

Additionally, $|T_p|$ = Fault (%age) * |T|

$$\text{Reliability} = \frac{|T| - |T_p|}{|T|} \quad (9)$$

In the event that one or more of the VM has a failure, the suggested reservation approach would ensure that all impacted tasks will complete their execution successfully by providing the reserved VM as an alternative VM, hence ensuring maximum system reliability in case of 50% of VMs will fail.

Proposed Pseudocode for the model:

Using the proposed ranked scheduling approach, the system effectively maps the incoming tasks with appropriate VMs. Further, the reliable fault-tolerant strategy that reserves the VMs for the pre-calculated R window that is estimated as the difference between the ST and CT also tracks the standards. The model makes sure that the tasks are completed successfully and achieves flexibility in the event any of the running VMs fail. The suggested algorithm for creating fault-tolerant scheduling is presented as follows.

The system's input parameters, including the Number and Length of tasks i.e., n(T) and $L(t_n)$, the Number and Capacity of VMs i.e., n(V) and $C(v_k)$ .and the available time of each VM were first set. Furthermore, the algorithm manages the matrix namely the Reservation matrix which denotes the reservation of the tasks.

i.e., ***Input****: n(T), $L(t_n)$, n(V), $C(v_k)$, $RT(v_k)$, RM[ ]*
Output *(Mapping (t_n, v_k), Reliability)*

1. Above all, the RT of all VMs is taken as zero indicating the VM is no load history.
2. The initialization of the RM matrix with the State flag includes the initialization of the chosen tasks, Start time, and Completion time with the computed Reservation window.
3. The task lengths range between 1 MI to 100 MI, and the machine capacity runs from 1 MIPS to 10 MIPS, in accordance with the model's low task and low machine heterogeneity assumptions.

Rank tasks on the basis of Task_id, Task_size//task ranking algorithms.
Do
Map the ranked tasks to the VMs. //allocation algorithm
While $\forall v_k s, (RT(v_k) = 0))$
Map task to VM having least RT.
Determine start time and completion time of task.
Adjust the revised RT for v_ks after each allocation.
Subtract the ST from the CT to determine the R window.
In order to continue processing the job in the event of failure, reserve VM in accordance with the computed R window.
Update RM[] with the most latest task data and set State = 1 to indicate that the computed AR window is reserved for the task.
For each allocation, repeat steps 4 through 14.
Verify the model's reliability once the task set has been finished.

4 Results and Discussions

The proposed model was evaluated on reliability by comparing it with other reliable existing models namely, FCWS, FR-MOS, and CWS. We selected five distinct states of task numbers with varying lengths for the simulation we created: Small(S)[n = 50 approx], Medium(M)[n=100 approx], Medium large(M-L)[n=200 approx], Large(L)[n = 400 approx], Extra large(E-L)[n=600 approx].

It can be seen from the depicted graph that the proposed model shows high reliability than all the considered models in all states afterward FCWS performs better. Furthermore, it is evident from the Fig. 2 that as the amount of tasks increases, the reliability of the considered approach decreases. However, as can be seen in the Improvement Percentage Table (Table 2), the suggested model exhibits a rise in the percentage of reliability improvement as the task count increases. This is because the suggested Model can efficiently handle various invoicing fault scenarios as the model is reserving the VMs for the dedicated window.

In S, the minimum improvement by the model was seen to be 0.30% while the maximum improvement was seen to be at 2.25%. In M, the minimum improvement by the model was seen to be 1.32% while the maximum improvement was seen to be 2.36%.

In M-L, the minimum improvement by the model was seen to be 1.53% while the maximum improvement was seen to be at 2.37%. In L, the minimum improvement by the model was seen to be 1.65% while the maximum improvement was seen to be at 2.18%. In E-L, the minimum improvement by the model was seen to be 1.26% while the maximum improvement was seen to be 2.45%.

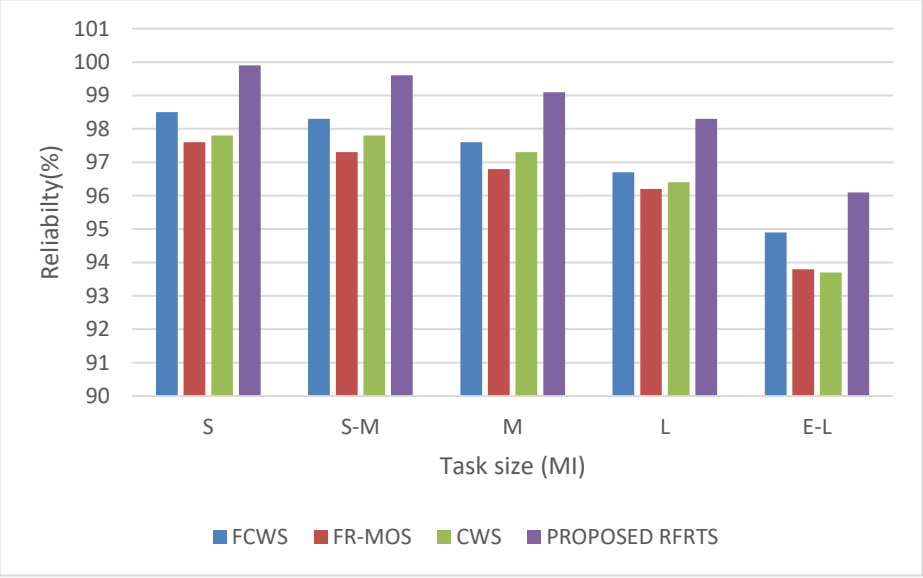

Fig. 2. Depiction of reliability in five considered task states.

Table 3. Comparative analysis of improvements in the proposed RFRTS.

FCWS	FR-MOS	CWS	Five states
0.30%	2.25%	2.04%	S
1.32%	2.36%	1.84%	M
1.53%	2.37%	1.84%	M-L
1.65%	2.18%	1.97%	L
1.26%	2.45%	2.56%	E-L

5 Conclusion and Future Directions

Since system reliability is one of the major issues in cloud systems, focusing on "execution till completion" is a crucial factor in enhancing reliability, therefore fault tolerance is necessary to achieve. The research suggests a method for task ranking by considering task lengths and task wait times. Besides, the algorithm implies an allocation strategy based on the determined rank value. Also, we provide a fault-tolerance strategy

in which VM reservations are made based on a pre-calculated reservation window. The paper's main focus is on the system's reliability. Moreover, it will unquestionably improve the task response times by focusing on the wait time of the tasks. The study's future plans demand for working with the suggested ranked scheduling technique, where the VMs will also be ranked to work over further optimizations of makespan and resource utilisation. The model will be extended by accompanying with load balancing technique for further optimization of the environment. The major drawback of the suggested allocation is that it could not minimize the makespan and there may be load imbalances over the VMs.

References

1. J. Sahni and D. Vidyarthi, "A Cost-Effective Deadline-Constrained Dynamic Scheduling Algorithm for Scientific Workflows in a Cloud Environment," IEEE Trans. Cloud Computing, vol. 6, no. 1, pp. 2-18, Jan.-March 2018.
2. J. He, K. Ota, M. Dong, L. T. Yang, M. Fan, G. Wang, and S. S. Yau, "Customized Network Security for Cloud Service," IEEE Trans. Services Computing, vol. 13, no. 5, pp. 801-814 , Sept.-Oct. 2020.
3. Z. Zhang, M. Dong, L. Zhu, Z. Guan, R. Chen, R. Xu, and K. Ota, "Achieving Privacy-Friendly Storage and Secure Statistics for Smart Meter Data on Outsourced Clouds," IEEE Trans. Cloud Computing, vol. 7, no. 3, pp. 638-649, July-Sept. 2019.
4. X. Zhou, G. Zhang, J. Sun, J. Zhou, T.Wei, and S. Hu, "Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based HEFT, Future Gen. Comput. Syst., vol. 93, pp. 278-289, Apr. 2019.
5. M. Farid, R. Latip, M. Hussin, and N. Hamid, "Scheduling Scientific Workflow Using Multi-Objective Algorithm With Fuzzy Resource Utilization in Multi-Cloud Environment," IEEE Access, vol. 8, pp. 24309-24322, Jan. 2020.
6. S. Kang, B. Veeravalli, and K. Aung, "Dynamic scheduling strategy with efficient node availability prediction for handling divisible loads in multi-cloud systems," J. Parallel Distrib. Comput. vol. 113, pp. 1-16, Mar. 2018.
7. D. Ardagna, M. Ciavotta, and M. Passacantando, "Generalized Nash Equilibria for the Service Provisioning Problem in Multi- Cloud Systems," IEEE Trans.
8. R. Li, Q. Zheng, X. Li, and Z. Yan, "Multi-objective optimization for rebalancing virtual machine placement," Future Gen. Comput. Syst., vol. 105, pp. 824-842, Apr. 2020.
9. Amazon EC2, http://aws.amazon.com/ec2/. Accessed Apr. 30, 2020.
10. Microsoft Azure, https://azure.microsoft.com. Accessed Apr. 25, 2020.
11. A. Dogan and F. • O zg• uner, "Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing," IEEE Trans. Parallel and Distributed Systems, vol. 13, no. 3, pp. 308-323, Mar. 2002.
12. L. Han, L. Canon, H. Casanova, Y. Robert, and F. Vivien, "Checkpointing Workflows for Fail-Stop Errors," IEEE Trans. Computers, vol. 67, no. 8, pp. 1105-1120, Aug. 2018.
13. A. Zhou, S. Wang, B. Cheng, Z. Zheng, F. Yang, R. N. Chang, M. R. Lyu, and R. Buyya, "Cloud Service Reliability Enhancement via Virtual Machine Placement Optimization," IEEE Trans. Services Computing, vol. 10, no. 6, pp. 902-913, Nov.-Dec. 2017.

14. H. Han, W. Bao, X. Zhu, X. Feng, and W. Zhou, ”Fault-Tolerant Scheduling for Hybrid Real-Time Tasks Based on CPB Model in Cloud,” IEEE Access, vol. 6, pp. 18616-18629, Feb. 2018.
15. Kalanirnika GR et al. Fault tolerance in cloud using reactive and proactive techniques. Int J Comput Sci Eng Commun. 2015;3(3):1159-1164.
16. Malik S, Huet F. Adaptive fault tolerance in real time cloud computing. Paper presented at: 2011 IEEE World Congress onServices (SERVICES); 2011:280–287.
17. Cheraghlou MN, Zadeh AK, HaghparastM. A survey of fault tolerance architecture in cloud computing. J Netw Comput Appl. 2016;61:81-92.
18. Zhao W, Melliar-Smith PM, Moser LE. Low latency fault tolerance system. Comput J. 2013;56(6):716-740.
19. Jhawar R, Piuri V, Santambrogio MD. A comprehensive conceptual system-level approach to fault tolerance in cloud computing. Paper presented at: 2012 IEEE International Systems Conference (SysCon);March 2012: IEEE:1–5.
20. Kochhar D, Kumar A, Hilda J. An approach for fault tolerance in cloud computing using machine learning technique. Int J Pure Appl Math. 2017;117(22):345-351.
21. Azaiez M, ChainbiW, Ghedira K. Hybrid fault tolerance model for cloud dependability. Paper presented at: International Conference on High Performance Computing and Communications; 2019:2436–2444.
22. Zheng Z, Zhou TC, Lyu MR, King I. FTCloud: a component ranking framework for fault-tolerant cloud applications. Paper presented at: 21st International Symposium on Software Reliability Engineering; 2010: IEEE:398–407.
23. Y. Wang, Y. Guo, Z. Guo, T. Baker, and W. Liu, ”CLOSURE: A cloud scientific workflow scheduling algorithm based on attack- Cdefense game model,” Future Gen. Comput. Syst., vol. 111, pp. 460-474, Oct. 2020.
24. Dogan, Atakan, and Fusun Ozguner. "Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing." *IEEE Transactions on Parallel and Distributed Systems* 13.3 (2002): 308-323.
25. X. Tang, K. Li, and G. Liao, ”An Effective Reliability-Driven Technique of Allocating Tasks on Heterogeneous Cluster Systems,” Cluster Computing, vol. 17, no. 4, pp. 1413-1425, Dec. 2014.
26. S. Kianpisheh, N. M. Charkari, and M. Kargahi, ”Reliabilitydriven scheduling of time/cost-constrained grid workflows,” Future Gen. Comput. Syst., vol. 55, pp. 1-16, Feb. 2016.
27. M. Farid, R. Latip, M. Hussin, and N. Hamid, ”Scheduling Scientific Workflow Using Multi-Objective Algorithm With Fuzzy Resource Utilization in Multi-Cloud Environment,” IEEE Access, vol. 8, pp. 24309-24322, Jan. 2020.
28. Tang, Xiaoyong. "Reliability-aware cost-efficient scientific workflows scheduling strategy on multi-cloud systems." IEEE Transactions on Cloud Computing 10.4 (2021): 2909-2919.