Journal of Scientific & Industrial Research
Vol. 83, December 2024, pp. 1317-1325
DOI: 10.56042/jsir.v83i12.7568

A Two-Phase Fault Tolerant System to Reduce the Service Level Agreement Violations

Archana Pandita[1], Prabhat Kumar Upadhyay[2]*, Sudhansu Kumar Mishra[2] & Nitish Kumar Ojha[3]

[1]Department of CSE, Amity University Dubai, UAE
[2]Department of EEE, Birla Institute of Technology, Mesra 835 215, Ranchi, India
[3]Department of CSE, Amity University, Noida 201 313, Uttar Pradesh, India

Received 26 December 2023; revised 18 July 2024; accepted 06 November 2024

Fault-tolerant systems are crucial in environments where uninterrupted service is mandatory and also in environments where continuous service is critical, such as in cloud computing and high-availability systems. The main goal of this research is to introduce a fault-tolerant framework that can effectively manage faults without leading to system failures, ensuring seamless service delivery. The proposed system follows a two-phase approach, incorporating fault detection and correction mechanisms that are designed to maintain system integrity and minimize performance degradation. Upon identifying a fault, the system deploys a correction technique aimed at preventing Service Level Violations (SLV), which could otherwise disrupt the service and breach Service Level Agreements (SLA). This fault-handling process is designed to mitigate interruptions while enhancing overall system performance. The proposed solution demonstrates a significant improvement in reducing SLA violations, achieving a reduction to 98.76%. Additionally, the efficiency of the system is enhanced through an increase in the task success rate, further ensuring the reliability of service. To validate its effectiveness, the performance of this fault-tolerant system has been benchmarked against other contemporary algorithms such as First Come First Serve (FCFS), Priority-based, and CD algorithms. The results reveal that the proposed approach offers notable improvements in response times, reducing them by 23%, 16%, and 33.1% respectively, in comparison to the FCFS, Priority-based, and CD algorithms. These results demonstrate the system's superior capability in handling faults and maintaining service quality, making it an ideal choice for environments where uninterrupted service is mandatory.

Keywords: Cloud computing, Fault detection, Machine learning, Scheduling, Service level violation

Introduction

One of the main challenges in cloud computing is task scheduling which is the process of assigning resources to task.[1] Scheduling techniques play a vital role in the efficient utilization of resources and maintaining effective throughput. Users register with Datacenter Broker (DB) which in turn, has Cloud Service Providers (CSP) registered with it. DB provisions the services for the users as per their requirement of resources. Cloud services are mainly classified as software, platform, and infrastructure.[2] Outlook Office 365 and Google Drive are the software-as-a-service, where entire infrastructure, data and applications are taken care of by the service provider. Google app engine and Azure are examples of platform-as-a-service where data and applications are looked after by the customer. Google compute engine, amazon etc. are the infrastructure-as-a-service, where along with data and application, runtime and middleware are also the responsibility of the customer.[3] CSP has many Data Centers (DC) spread geographically, and as data centers have grown complex, enterprises have started adopting server virtualization technologies, including cluster servers, Network Interface Card (NIC) teaming and Virtual Machine Ware (VMW).[4–6]

This complexity and dynamicity give rise to errors which are considered to be the primary source of faults resulting in system failure. Cloud services comprise various service components which are in general, geographically scattered. To guarantee the continuous availability of cloud service, it's very important to monitor the service components and detect possible faults. Several possible reasons for fault have been reported in the literature which include omission fault that can occur because of disk full or denial of service, ageing faults that can be because of early or late faults, and lifecycle faults that can be caused by versioning or service expiry.[7,8]

A service level agreement is a contract between the cloud service providers and cloud customers. It's a

*Author for Correspondence
E-mail: uprabhatbit@gmail.com

document which has all the terms and conditions related to the quality of services that are mutually agreed upon by both parties. Any breach in the contract is known as Service Level Violation (SLA) and the breaches can result in penalties. There are several reasons which result in the breach, such as a mismatch between the requested and assigned resources or a degradation in the performance caused by any fault, failure or malfunction of a component in the system. Fault-tolerant approaches are of paramount importance to avoid faults and hence any violation of the service level agreement. Integration of scheduling with fault-tolerant approaches is considered as one of the most critical factors in designing cloud-scheduling strategies with fewer SLA violations.[9] The role of a cloud scheduler is to optimally use the available resources to eliminate the chances of service-level violations. However, resources are subject to failure, which has a high impact on the provisioning, credibility, and performance in a cloud infrastructure. Therefore, there is a need for a system in the cloud, which can identify faults, correct them, and guide the scheduler to migrate the virtual machine in the best possible way. The property of a system to keep performing even in the case of a fault is termed as fault tolerance which handles the faults gracefully without affecting the end-user and without violating the SLA. The faults can be mainly classified as Crash fault and Byzantine fault. In a crash fault, the functioning of the system halts whenever the breakdown occurs whereas, in the case of a Byzantine fault, the elements of the system act randomly when the breakdown occurs, which results in an inaccurate output. In both cases, the system performs arbitrarily and hence doesn't give an appropriate output. To make a system fault-tolerant, it should be able to identify faults, predict them and recover from them in such a way that the overall system behaves as expected and gives an appropriate output in terms of the time taken and resources utilized. Fault tolerance can be categorized as reactive, proactive, or adaptive. Reactive fault tolerance lets the fault occur, and once it has occurred and is detected, necessary measures are taken, whereas a proactive technique takes actions well before the fault has occurred. Techniques like software rejuvenation, self-healing, load balancing, and checkpointing are proactive, whereas techniques like replication, job migration, rescue overflow, and task resubmission are reactive. Adaptive techniques such as replication and fragmentation continuously monitor the system, and when required, take necessary action.[10] There have been different fault tolerance parameters, which need to be considered while designing a scheduling technique. The parameters include availability, response time, reliability, usability, cost, scalability etc. Researchers in their studies have focused on a different set of parameters and put forward various frameworks using fault tolerance tools, which include Hadoop, Shelp, HAproxy and Amazon EC2.[11–13]

A plethora of research is going on to achieve fault tolerance in cloud computing which in turn helps to achieve lesser violations in the service level agreement, and concerning which, many fault-tolerant models have been proposed in the literature. Dynamic Hyper cubic peer-to-peer cloud (HPCloud) applies the MapReduce technique and improves response time but lacks the dynamicity.[14] Non-dominated Sorting Genetic Algorithm (NSGA)-II is based on the Pareto dominance relationship and is found to be efficient for small and medium-sized scheduling problems with a lack of heterogeneity and dynamicity.[15] The Dynamic and reliability driven real-time fault-tolerant scheduling algorithm (DRFACS) uses an active replication technique.[16] A significant increase in reliability and Quality of Service (QoS) is noticed in DRFACS, but the critical tasks were not prioritized. Heterogeneous earliest finish time is a task replication technique which attains good fault tolerance but is observed to be deficient in dynamicity.[17] Another fault-tolerant method known as the hybrid heuristics scheduling approach incorporates the migration of Virtual Machine (VM) in its algorithm.[18] The author claims that this method is better than Min-Min, Min-Max, MCT, PSO and GA in terms of execution time and makes pan but not all fault-tolerant parameters have been analyzed. However, min-min based time and cost trade-off shows good results for real-life workflows.[19] The performance, power and failure aware relaxed time task execution model uses a slowdown estimator and accomplishes SLAs, but there is a trade-off between fault tolerance and performance.[20] Energy-aware fault-tolerant scheduling minimizes the overall failure rate by 50% and saves a total energy of 76%, but the overall system performance is compromised.[21]

The machine learning approach is based on improving a system based on prior information. Machine learning has been found to be promising for

scheduling problems by many researchers. The ensemble of the Bayesian model, a failure detection method put forward by Qiang *et al.*, finds irregular activities and then applies the decision tree classifier for the prediction of faults based on supervised learning.[22] Machine learning and other soft computing techniques have also been used by several researchers, to address the problems related to scheduling, and they also have put forward illustrations of effective incorporation of machine learning in scheduling systems.[23–26] Addressing the challenges like effective utilization of virtual instances security and communication, about applying a machine learning algorithm on vast volumes of data, authors have proposed a basic scheme and an advanced scheme using deep learning concepts.[27] In the basic scheme, authors have used M-FHE whereas in the advanced scheme, authors have applied the double encryption mechanism together with fully homomorphic encryption. Distributed GraphLab presented by Yucheng *et al.*, is a framework for machine learning in the cloud.[28] The development is based on data versioning and pipelined locking. The fault-tolerance model following this approach was developed using the classic Chandy-Lamport snapshot algorithm.[29] System overhead and execution time were also predicted using machine learning techniques by a few researchers.[30]

Several studies have focused on software faults prediction using techniques like fuzzy logic, Naïve Bayes, neural network etc.[31–33] Menzies *et al.* applied the data mining algorithm on the public NASA dataset and evaluated the results using a false alarm and detection probability.[34] Koru and Liu used a similar dataset to construct the fault prediction model, and the performance metrics are evaluated using F-measures.[35] The Random Forest machine learning technique was used by Catal et al. to evaluate the FT model in terms of accuracy and by Ambikavathi *et al.* to develop predictor selection and attack classification for Intrusion detection.[36,37] Ping Li *et al.* proposed Privacy-preserving outsourced classification in cloud computing, a framework to preserve the confidentiality of data.[38] Several other studies focus on performance Evaluation of the Cloud-based QR Code Identity Tag System with Cloudlets.[39]

Though many researchers have put forward different models in their research work, there are many causes which hinder the efficiency of the system and need to be addressed properly to get rid of imperfect scheduling strategies. Service Level violation occurs when there is a breach of the agreement between the service provider and the customer which may be caused by degradation in the performance of virtual machines due to any system failure. This results in a substantial penalty to be paid by the service provider, and the customer has to compromise on Quality of Service. Hence, the timely detection and correction of a fault in the system is of paramount importance. The following are the contribution of this paper:

I. The faults are successfully detected and corrected.

II. The success rate of the task is increased substantially.

III. The SLA violation and response time is reduced.

System Model

The system model proposed in this work is shown in Fig. 1 and the major components – scheduler, monitoring and analytics used are elaborated below:

Scheduler: Allocating the incoming task to a Virtual Machine (VM) is the responsibility of the scheduler. It has all the information about the machines, and it matches the incoming task to a corresponding VM. Load balancing is also performed by the scheduler so that the load can be transferred from overloaded machines to under loaded ones. Every VM maintains a queue of the incoming tasks and the length of the queue represents the load on a particular VM.

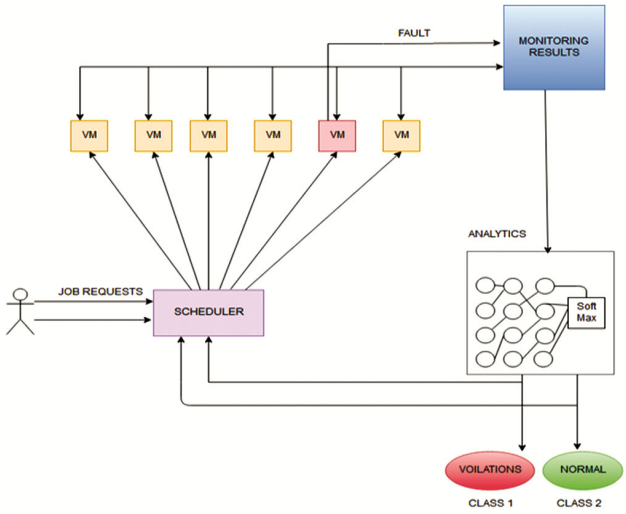

Fig. 1 — Proposed System Model for detection and correction of faults

Monitoring Module: A monitoring module monitors the VM's performance. The monitoring results are logged into the time-series database and the database can be accessed by the scheduler. This logged data is then used to identify the VM's performance. If the performance of a particular VM is degraded beyond a certain pre-defined threshold, it is assumed to be a faulty VM.

Analytics Module: For each new request, the scheduler uses the analytics module to evaluate the probability of fault for each VM for a given task. To achieve this, the analytics module uses neural network equipped with Softmax function. The decision taken by the analytics is based on SLA violation probabilities.

In the analytics module, initially, random weights were assigned to each neuron during training and net weighted input is computed. The output after being squashed between 0 and 1 is allowed to pass on to the next layer, which in turn, restructures the weights of the neurons. After performing an iterative approach, the exact weight for each neuron is identified.

Detection and Correction of Fault

Let C = $\{D_1, D_2, D_3, D_4 \ldots\ldots D_n\}$, where C is the cloud and D_n is the number of data centers located in n locations. Each data centre constitutes a pool of virtual machines with varying sizes. Given a set of inter virtual machines distance (latency) $\{V_{ab}\}$, the first objective is to find the permutation matrix X, in order to minimize the following function:

$$P(M) = \frac{1}{2}\sum_{a=1}^{N}\sum_{b=1}^{N}\sum_{i=1}^{N} V_{ab} M_{ai} M_{b(i+\theta)} \quad \ldots (1)$$

where, V_{ab} represents the latency between location 'a' and 'b'. M represents the permutation matrix with rows as locations and columns as timeslots, and N is the number of locations where any VM could reside. The notation I + θ indicates whether the VM is visited or not. Considering the above as a Travelling Salesman Problem (TSP) with the constraint not to revisit any VM, this constraint is enforced by applying Lagrange multipliers and a barrier function. To further process the request, the scheduler uses the output produced by the analytics module, and in the event of a fault, VM migration takes place. Identifying an alternative VM for the assignment of tasks, enforces the two-way Winner-takes-all (WTA) constraint. The typical WTA problem is to select the highest-ranked VM from the input K from a set of probable inputs E and is represented as:

$$f(r_i) = \begin{cases} 1, & if\ r_i \in K\ with\ highest\ rank\ in\ E \\ 0, & otherwise \end{cases} \quad \ldots (2)$$

where, r is the rank of E. Softmax regression, also known as multinomial logistic regression, is a generalization of logistic regression and is applicable in handling multiple classes. The label y can produce k different probability values. In the case of VM migration in the event of a fault, Softmax is used to generate the probability value for each VM's SLA violations. Therefore, in our test input, the hypothesis would be to calculate the probability of SLA violations for each VM. Given the test input x,

$$P(y = \frac{k}{x}) = \begin{bmatrix} p\left(y = \frac{1}{x}\right) \\ p\left(y = \frac{2}{x}\right) \\ \cdot \\ \cdot \\ \cdot \\ p(y = \frac{K}{x}) \end{bmatrix} \quad \ldots (3)$$

The hypothesis is to estimate the class label SLA violation and SLA preserving. Thus, the output is a 2-dimensional matrix as shown in equation 4, given that the sum of the elements would be equal to 1.

$$Z^L = w^L\, a^{[L-1]} + b^{[L]} \quad \ldots (4)$$

where, 'a' is the activation function of the previous layer, and 'b' is the bias of the final layer. Let 't' be the temporary value for the activation function in Softmax regression.[40] Mathematical expression is given in equation 5 as:

$$t = e^{Z[L]} \quad \ldots (5)$$

Therefore, to normalize the distribution, so that it sums to one,

$$a^{[L]} = \frac{e^{z[L]}}{\sum_{j=1}^{d} t_i} \quad \ldots (6)$$

Further,

$$a_i^{[L]} \frac{t_i}{\sum_{j=1}^{d} t_i} \quad \ldots (7)$$

where, a_1, a_2, a_3, ..a_d are the parameters and d is the total number of parameters. To implement Softmax regression, it is usually convenient to represent $a^{[L]}$ as a n-by-k matrix obtained by concatenating $a^{(1)}$, $a^{(2)}$,…,$a^{(K)}$ into columns.

Performance Metrics and Experimental Setup

Performance metrics such as the success ratio, response time and SLA violations have been used to

investigate the effectiveness of the algorithm under consideration. Success Ratio (Sr) has been calculated for all the jobs completed over the total jobs allocated. Another performance metric, Response Time (R) which is one of the most generic metrics to evaluate the performance of any scheduling algorithm, is calculated as the amount of the jobs N, executed per unit of time T. Similarly, SLA violations, which are the number of observed violations that occurred over several jobs allocated to a system, are computed.

This proposed algorithm uses the dataset obtained from the archive and is implemented in Cloudsim toolkit version 3.0.[41,42] To establish priority among tasks, jobs are divided into five priorities where each job is associated with one priority. Here, 1 refers to the highest priority job and 5 is considered as the least. The specifications of each datacenter configuration are given in Table 1. In this setup, six data centers were deployed in different locations with 20 different users' locations to send job requests in uniform distribution as well as the Poisson distribution manner.

Furthermore, to introduce the faults, a random virtual machine is put into sleep mode. The data of the virtual machine is stored, and the same environment is regenerated for other scheduling algorithms to evaluate the performance of the system. The experiment has been performed in three different experimental scenarios as presented in Table 2.

A job execution failure may be caused due to network failure, hardware failure or overload conditions. In this paper, we presented a classifier using the Softmax function to schedule the assigned jobs. This technique allows the scheduler to predict the probability of a job failure if assigned to a particular resource. Continuous monitoring of virtual machines helps the scheduler to know the status of the resources and for each new job request, the probability is calculated for a certain resource and the job is submitted for execution accordingly. The proposed system has shown its capability to find a near-perfect solution and finally an optimal solution.

Table 1 — Configuration of a Datacenter

Configuration of a Datacenter	
Hypervisor	Xen
RAM (mb)	204800
BW	10000000
Number of processors	4
Processor speed	10000
Arch	x86
Cost/VM/Hr ($)	0.05
User grouping factor	1000
Request grouping factor	100
Executable instructions length per request	250
Host Memory	2048
Storage Capacity	100000000
Bandwidth	100000
Task Size (cloudlets)	10 to 40 MB
Task Frequency	10000/minute/user
Fault frequency	40 secs/VM

Results and Discussion

In the proposed model, the requests are submitted to the scheduler, which in turn relates to the analytics module. The analytics module uses the existing logged data stored in the time-series database by the monitoring tool. The analytics module checks for any degradation in the performance of any virtual machine, which may be possible due to the probable presence of a fault in the system. Inferences are deduced from the output stored, which helps in further decision-making. The Analytics module is equipped with a neural network for deducing the output in terms of SLA violation. The classification is done based on SLA violations with discrete values 0 (no) and 1(yes). The features of a virtual machine such as latency, queue length, etc. were stored and the response time was calculated. Having calculated the response time, the SLA violation is determined by subtracting it from the expected time of completion. Softmax regression is used to calculate the SLA violation probability of the VM targeted for the next request. If the SLA probability is found to be greater than the threshold, the job is migrated to another VM. To evaluate the performance of the proposed model, a comparison is also made with the existing well-known algorithms-First Come First Serve (FCFS), Priority-Based Algorithm (PBA) and Contrastive Divergence

Table 2 — Experimental Scenarios

Scenario	Number of VM per datacenter	Task frequency (minute/user)	Number of VM with faults	Fault frequency (seconds /VM)
Scenario 1	200	10000	500	40
Scenario 2	200	15000	700	50
Scenario 3	200	22000	Nil	Nil

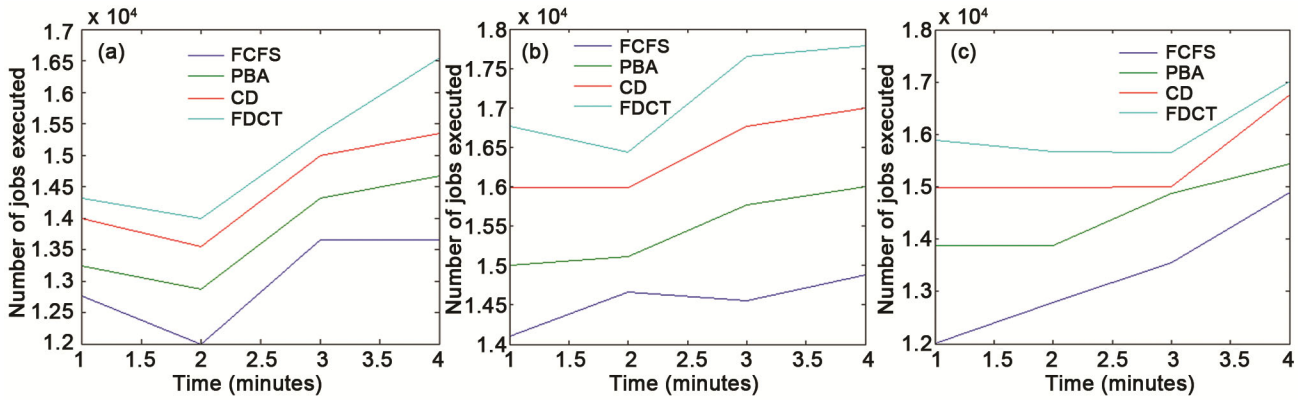

Fig. 2 — Comparison among different scenarios for number of jobs executed: (a) Scenario 1, (b) Scenario 2, (c) Scenario 3

(CD) with the help of experiments undertaken in 3 different scenarios where the scenarios differ in the number of VMs, task frequency, number of VMs with fault and fault frequency. In each scenario, a fixed number of jobs are submitted to each algorithm for execution. The results, as shown in Fig. 2(a-c), reveal that the proposed algorithm executes more jobs than non-machine learning algorithms such as FCFS, PBA, and even a machine learning algorithm CD, in all three scenarios.

The proposed model initially shows a decline in the number of jobs getting executed. But after a gap of 2–3 minutes, it started executing more jobs. It may be because, during online training, a certain initial time is required to train the system and after sufficient training, the algorithm can classify the job as per the availability of the virtual machines with the least probability of SLA violations. The proposed model outperforms other machine learning algorithms such as CD, as well as non-machine learning algorithms, FCFS and PBA in all three scenarios in terms of the number of jobs executed.

A comparative result is displayed in Table 3 considering the success ratio of all the algorithms. Machine learning algorithms were found to perform better than non-machine learning algorithms in terms of the number of jobs completed over the total jobs submitted. Also, it is clear that the proposed algorithm, in totality, outperforms other existing algorithms and records the highest success ratio of 0.98 in scenario 3 followed by 0.92 in scenario 1. PBA recorded the lowest success ratio in all three scenarios.

A comparative response time graph is presented in Fig. 3, which supports the fact that the proposed algorithm, on average, takes 23%, 16% and 33.1% less time in all three scenarios, as compared to the CD

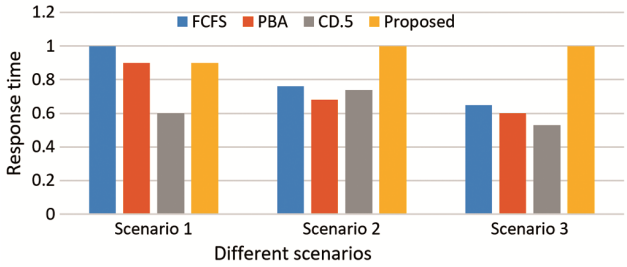

Fig. 3 — Response time comparison

Table 3 — Success ratio of tasks for three scenario

Scenario	FCFS	PBA	CD	Proposed
1	0.76	0.69	0.89	0.92
2	0.68	0.64	0.86	0.90
3	0.84	0.79	0.93	0.98

Table 4 — SLA violations (in %)

Scenario	FCFS	PBA	CD	Proposed
1	10.1	20.32	5.20	2.43
2	13.3	26.23	6.30	3.12
3	9.2	17.43	3.21	2.31

algorithm. However, in Scenario 1, the performance of FCFS is observed to be marginally better, which may be due to the relatively low number of jobs resulting in a lesser load on each of the faulty machines. It can also be observed from the results obtained that with the increase in job requests machine learning-based algorithms outperform traditional ones. In scenarios 2 and 3, since the number of job requests is very high; the proposed algorithm is observed to be ahead of others in terms of response time.

Minimized SLA violation in percentage as presented in Table 4 reflects the fact that PBA is the worst performer amongst all with 21% more violations as compared to FCFS. Also, the performance of the proposed algorithm intends to

minimize the SLA violation with an effectiveness of 98.76%, which is further followed by the performance of CD algorithm (96.3%).

The reason for the minimization is likely to be its inherent potential of assigning the task to the right virtual machine and the classification ability to identify a virtual machine with the probability of least SLA violation.

Fault-tolerant systems have been a topic of significant interest in both academic research and industrial applications, with numerous strategies proposed to ensure system resilience. The results of the proposed two-phase fault-tolerant system align with and advance previous studies in fault tolerance within cloud computing environments. For example, Kumar *et al.* discussed fault-tolerant scheduling algorithms aimed at minimizing Service Level Agreement (SLA) violations in cloud environments, reporting SLA violation reductions around 95%, while the proposed system achieves a further reduction to 98.76%.[43] This improvement can be attributed to the efficient correction techniques employed in the proposed approach. Furthermore, the performance comparisons between the proposed system and other algorithms such as First Come First Serve (FCFS), Priority-based, and CD algorithms show significant improvements. Another researcher examined FCFS-based fault-tolerant mechanisms, but they noted that such algorithms often result in longer response times due to their simplistic task-handling nature.[44] Similarly, another researcher highlighted the limitations of priority-based algorithms, which, while more efficient than FCFS, still struggle with optimal fault handling when task prioritization is inconsistent with system dynamics.[45] In contrast, the proposed system reduces response times by 23%, 16%, and 33.1%, respectively, in comparison to these algorithms, indicating a more efficient fault detection and recovery mechanism. The correction techniques implemented in the proposed system also align with strategies highlighted in existing who advocated for proactive fault management approaches in distributed systems.[46] However, the proposed system improves upon their reactive fault recovery method by integrating a more refined two-phase fault management system, resulting in both improved task success rates and reduced response times. These results suggest that the proposed approach contributes meaningfully to existing literature on fault tolerance and sets a new benchmark in terms of SLA violation reductions and performance optimization.

Conclusions

In this paper, we have addressed the issue of fault tolerance in cloud computing and proposed a two-phase fault detection and correction technique. In high-reliability systems, fault tolerance is essential for ensuring continuous operation and minimizing downtime. This paper outlines a two-phase fault-tolerant system designed to first detect faults and subsequently correct them. The proposed system ensures high availability and reliability through efficient fault management. The proposed algorithm outperforms the traditional algorithms and other machine learning algorithms. The Least SLA violation is recorded with the least response time of tasks assigned. As an efficient system is established by reducing the time consumption of workloads, the system can work in accordance with the SLA. More tasks are found to be executed successfully using the proposed algorithm, thus increasing the success rate. The results are quite encouraging and can be applicable in developing an automated system for detecting and correcting possible failures. Also, as a part of the future plan, we would like to investigate the overhead computation considering the real scenario of the cloud user's requests.

Competing Interest

The authors declare that they have no competing interests.

References

1 Saidi K & Bardou D, Task scheduling and VM placement to resource allocation in Cloud computing: challenges and opportunities, *Cluster Comput*, **26(5)** (2023) 3069–3087.

2 Ali Z, Mahmood A, Khatoon S, Alhakami W, Ullah S S, Iqbal J & Hussain S, A generic Internet of Things (IoT) middleware for smart city applications, *Sustainability*, **15(1)** (2022) 743.

3 Cérin C, Coti C, Delort P, Diaz F, Gagnaire M, Gaumer Q, Guillaume N, Lous J, Lubiarz S, Raffaelli J & Shiozaki K, Downtime statistics of current cloud solutions, *IWGCR*, *Tech Rep*, **1(2)** (2013) 130.

4 Katal A, Dahiya S & Choudhury T, Energy efficiency in cloud computing data centers: a survey on software technologies, *Cluster Computing*, **26(3)** (2023) 1845–1875.

5 Singh S & Chana I, A survey on resource scheduling in cloud computing: Issues and challenges, *J Grid Comput*, **14(2)** (2016) 217–264.

6 Li Y, Chen M, Dai W & Qiu M, Energy optimization with dynamic task scheduling in mobile cloud computing, *IEEE Syst J*, **11(1)** (2017) 96–105.

7 Rahimpour S, Tarzamni H, Kurdkandi N V, Husev O, Vinnikov D & Tahami F, An overview of lifetime management of power electronic converters, *IEEE Access*, **10** (2022) 109688–109711.

8 Latiff M S, Madni S H & Abdullahi M, Fault tolerance aware scheduling technique for cloud computing environment using dynamic clustering algorithm, *Neural Comput Appl*, **29(1)** (2018) 279–293.

9 Son S & Jun S C, Negotiation-based flexible SLA establishment with SLA-driven resource allocation in cloud computing, in 2013 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (IEEE) 2013, 168–171.

10 Furht B & Escalante A, *Handbook of Cloud Computing*, (Springer, New York), **3** (2010), https://doi.org/10.1007/978-1-4419-6524-0.

11 Rehman U, Aguiar R L & Barraca J P, Fault-tolerance in the scope of cloud computing, *IEEE Access*, **10** (2022) 63422–63441.

12 Mohammadian V, Navimipour N J, Hosseinzadeh M & Darwesh A, Fault-tolerant load balancing in cloud computing: a systematic literature review, *IEEE Access*, 10 (2022) 12714–12731, https://doi.org/10.1109/ACCESS.2021.3139730.

13 Tawfeeg T M, Yousif A, Hassan A, Alqhtani S M, Hamza R, Bashir M B & Ali A, Cloud dynamic load balancing and reactive fault tolerance techniques: a systematic literature review (SLR), *IEEE Access*, 10 (2022) 71853–71873, https://doi.org/10.1109/ACCESS.2022.3188645.

14 Gangeshwari R, Subbiah J, Malathy K & Miriam D D, Hpcloud: a novel fault tolerant architectural model for hierarchical mapreduce, *Proc ICRTIT* (*IEEE*), (2012) 179–184.

15 Li H & Zhang Q, Multiobjective optimization problems with complicated Pareto sets, MOEA/D and NSGA-II, *IEEE Trans Evol Comput*, 13(2) (2009) 284–302, https://doi.org/10.1109/TEVC.2008.925798.

16 Qin X & Jiang H, Dynamic, reliability-driven scheduling of parallel real-time jobs in heterogeneous systems, *Proc Int Conf Parallel Proces* (IEEE) 2001, 113–122.

17 Zhao H & Sakellariou R, An experimental investigation into the rank function of the heterogeneous earliest finish time scheduling algorithm, *Proc European Conf Parallel Proces* (Springer, Berlin, Heidelberg) 2003, 189–194.

18 Tsai C W, Huang W C, Chiang M H, Chiang M C & Yang C S, A hyper-heuristic scheduling algorithm for cloud, *IEEE Trans Cloud Comput*, 2(2) (2014) 236–250, doi: 10.1109/TCC.2014.2315797.

19 Garg S K, Buyya R & Siegel H J, Time and cost trade-off management for scheduling parallel applications on utility grids, *Fut Gener Comput Syst*, 26(8) (2010) 1344–1355.

20 Sampaio A M & Barbosa J G, Dynamic power-and failure-aware cloud resources allocation for sets of independent tasks, *In: 2013 IEEE Int Conf Cloud Eng* (IEEE) 2013, 1–10.

21 Gao Y, Gupta S K, Wang Y & Pedram M, An energy-aware fault-tolerant scheduling framework for soft error resilient cloud computing systems, in *Proc Conf Design, Automat Test Europe*, *European Design and Automat Associat* (IEEE) 2014, 94.

22 Guan Q, Zhang Z & Fu S, Ensemble of Bayesian predictors and decision trees for proactive failure management in cloud computing systems, *J Commun*, 7(1) (2012) 52–61, https://doi.org/10.4304/jcm.7.1.52-61.

23 Li S, Yu T, Cao X, Pei Z, Yi W, Chen Y & Lv R, Machine learning-based scheduling: a bibliometric perspective, *IET Collaborat Intel Manuf*, 3(2) (2021) 131–146, https://doi.org/10.1049/cim2.12004.

24 Groth M, Schumann M & Nickerson R C, Characteristics of production scheduling problems in the era of Industry 4.0 – a review of machine learning algorithms for production scheduling, in *Int Conf Flex Automat Intel Manuf* (Springer, Cham) 2024, 119–127.

25 Waubert de Puiseau C, Meyes R & Meisen T, On reliability of reinforcement learning-based production scheduling systems: a comparative survey, *J Intell Manuf*, 33(4) (2022) 911–927, https://doi.org/10.1007/s10845-022-01915-2.

26 Abdelaziz A, Elhoseny M, Salama A S & Riad A M, A machine learning model for improving healthcare services on cloud computing environment, *Measurement*, 119 (2018) 117–128.

27 Li P, Li J, Huang Z, Gao C Z, Chen W B & Chen K, Privacy-preserving outsourced classification in cloud computing, *Cluster Comput*, 21 (2018) 277–286, https://doi.org/10.1007/s10586-017-0849-9.

28 Low Y, Bickson D, Gonzalez J, Guestrin C, Kyrola A & Hellerstein J M, Distributed GraphLab: a framework for machine learning and data mining in the cloud, *J Proc VLDB Endowment*, 5(8) (2012) 716–727.

29 Carbone P, Fóra G, Ewen S, Haridi S & Tzoumas K, Lightweight asynchronous snapshots for distributed dataflows, *arXiv preprint*, arXiv:1506.08603 (2015).

30 Dean D J, Nguyen H & Gu X, UBL: unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems, in *Proc 9th Int Conf Autonom Comput* (ACM) 2012, 191–200.

31 Chen Z, Zhu Y, Di Y & Feng S, Self-adaptive prediction of cloud resource demands using ensemble model and subtractive-fuzzy clustering-based fuzzy neural network, *Comput Intell Neurosci*, 2015.1 (2015) 919805.

32 Dai Y, Xiang Y & Zhang G, Self-healing and hybrid diagnosis in cloud computing, in *IEEE International Conference on Cloud Computing* (Springer, Berlin, Heidelberg) 2009, 45–56, https://doi.org/10.1007/978-3-642-10665-1_5.

33 Thwin M M & Quah T, Application of neural networks for software quality prediction using object-oriented metrics, in *Proc 19th Int Conf Softw Mainten* (Amsterdam, The Netherlands) (IEEE) 2003, 113–122.

34 Menzies T, Greenwald J & Frank A, Data mining static code attributes to learn defect predictors, *IEEE Tran Softw Eng*, 33(1) (2007) 2–13, https://doi.org/10.1109/TSE.2007.256941.

35 Mehmood I, Shahid S, Hussain H, Khan I, Ahmad S, Rahman S & Huda S, A novel approach to improve software defect prediction accuracy using machine learning, *IEEE Access*, 11 (2023) 63579–63597, doi: 10.1109/ACCESS.2023.3287326.

36 Catal C & Diri B, Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem, *Inform Sci*, 179(8) (2009) 1040–1058, https://doi.org/10.1016/j.ins.2008.12.001.

37 Ambikavathi C, Srivatsa S & Krishna S K, Predictor selection and attack classification using Random Forest for intrusion detection, *J Sci Ind Res*, 79(5) (2020) 349–355.

38 Li P, Li J, Huang Z, Gao C Z, Chen W B & Chen K, Privacy-preserving outsourced classification in cloud computing,

Cluster Comput, 21 (2018) 277–286, https://doi.org/10.1007/s10586-017-0849-9.

39 Uzun V, Performance evaluation of the cloud-based QR code identity tag system with cloudlets, *J Sci Ind Res*, 79(12) (2020) 1101–1106.

40 Softmax regression, http://deeplearning.stanford.edu/tutorial/supervised/SoftmaxRegression/ (Accessed December 2024)

41 European Union Open Data Portal, Dataset: https://data.europa.eu/euodp/data/dataset/yUBHDpCh8MDqL9Gub8Qmq /resource/f358fa78-d6fa-42be-bc9f-ee28669a0543, (Accessed January 2024)

42 CloudSim: A framework for modelling and simulation of cloud computing infrastructures and services, *Cloud Computing and Distributed Systems (CLOUDS) Laboratory*, School of Computing and Information Systems, The University of Melbourne, Australia, http://www.cloudbus.org/cloudsim/ (Accessed January 2023)

43 Kumar R, Sharma A & Patel P, Fault-tolerant scheduling algorithms in cloud computing environments: a comprehensive review, *J Cloud Comput: Adv Syst Appl*, 9(1) (2020) 14–32, https://doi.org/10.1186/s13677-020-00205-7.

44 Patel S, Gupta R & Singh S, First come first serve based fault-tolerant mechanisms: Analysis and optimization, *Int J Comput Appl*, 179(9) (2019) 15–25, https://doi.org/10.5120/ijca2019919453.

45 Singh V & Kaur R, Priority-based fault tolerance in cloud computing: challenges and solutions, *IEEE Trans Cloud Comput*, 10(4) (2021) 2057–2067, https://doi.org/10.1109/TCC.2021.3084971.

46 Zhao J, Li X & Wang H, Proactive fault management in distributed systems: Techniques and approaches, *ACM Comput Surveys*, 50(3) (2018) 40–58, https://doi.org/10.1145/3197681.