International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Dynamic Fault Tolerance Management Algorithm for VM Migration in Cloud Data Centers

Bikash Chandra Pattanaik[1*], Bidush Kumar Sahoo[2], Bibudhendu Pati[3], Suprava Ranjan Laha[4]

Submitted: 22/04/2023 **Revised**: 24/06/2023 **Accepted**: 03/07/2023

Abstract: Fault tolerance is critical in constructing robust cloud computing systems to ensure uninterrupted service delivery and maintain economic benefits despite potential faults. This paper presents a novel layered modeling architecture that combines reactive and proactive fault modeling theories to enable reliable, survivable cloud-based applications by addressing fault tolerance concerns. This paper examines the issues of dynamic fault tolerance management and virtual machine (VM) migration in cloud data centers. We introduce a comprehensive algorithm that efficiently manages fault tolerance through proactive measures by leveraging a layered modeling architecture. The algorithm considers defect prediction and resource allocation techniques to minimize service interruptions and maximize resource utilization. It incorporates reactive and proactive fault modeling to identify and respond to faults, anticipates potential faults, and takes preventative measures. This integration makes the cloud computing environment more robust and reliable. However, extensive simulations and evaluations demonstrate the proposed algorithm's effectiveness in reducing service downtime, ensuring application reliability, and sustaining optimal performance. The algorithm's ability to dynamically migrate virtual machines (VMs) based on defect prediction contributes to efficient resource allocation and load balancing, mitigating potential bottlenecks and enhancing system resilience. The results demonstrate the applicability and effectiveness of the proposed framework for maintaining cloud-based applications' dependability. Combining reactive and proactive fault modeling theories, the proposed algorithm provides a comprehensive method for keeping cloud-based applications reliable and fault-tolerant.

Keywords: *fault tolerance, load balancing, reactive fault tolerance, proactive fault tolerance, resource utilization, cloud computing, dynamic VM migration, resource allocation.*

1. Introduction

Cloud computing enables the sharing of resources on a pay-per-use basis [1]. Cloud services must ensure that resources are available when they are needed [2]. In-cloud resources are often insufficient to support complex jobs [3]. This problem becomes acute when the work load grows and submissions increase. In cloud computing, strategies are devised to address the problem of insufficient resources. Recent strategies have included reserving. When resources become available, reservation guarantees that jobs are added to the system as soon as possible [4]. This prevents as sudden increase in load on the server and virtual machine, which can cause failures when more jobs are added. A broker is responsible for managing the nodes in the cloud architecture& determining the needs of each cloudlet [5]. Cloudlets that match the desired requirements are added to the list, while those that do not are discarded. This method reduces throughput by reducing the number of cloudlets that match the desired configuration. Cloud computing is a recent development that allows a client to access any resources needed at any time. Cloud services allow users to perform a lot of tasks at the same time. However, these services should be highly reliable and stable so that they can satisfy the requirements of users. Below are these layers: At the top of the structure are cloud applications where clients send their applications. Operating systems and application frameworks reduce VM burden in the platform layer below the application layer. Infrastructure components like storage & networking are in the virtualization layer. Cloud computing services allow companies to reduce the cost of building and maintaining a computing environment by using a cloud provider's services. The benefits of using clouds, such as unlimited data storage and object computation, also make them popular among businesses. For data- and compute-intensive applications like those used in scientific research, cloud computing can be a cost-effective alternative since it allows users to complete computation activities on a pay-as-they-go basis without the hassle of creating and handling its cloud strategies. Large-scale cloud computing systems are especially prone to malfunction. Both Daniel B. Stewart [6] and Michael J. Howard [7] detail the unwanted cloud strategies of 2013 and 2014, correspondingly. Some users experienced difficulties

[1] *Gandhi Institute for Education and Technology, Affiliated to Biju Patnaik University of Technology Rourkela, Odisha, India*
ORCID ID: 0009-0000-6713-1492
[2] *GIET University, Gunupur, Odisha, India*
ORCID ID: 0000-0002-5044-0819
[3] *Ramadevi Women's University, Bhubaneswar, Odisha, India*
ORCID ID: 0000-0002-2544-5343
[4] *Siksha 'O' Anusandhan University, Bhubaneswar, Odisha, India*
ORCID ID: 0000-0003-1847-8419
Corresponding Author Email: bikashpatnaik73@gmail.com

accessing popular cloud services including Facebook, Amazon, and Google Drive. As roughly services of customers are absent entirely or in part for some time due to these outages, clouds lost data, money, and customer trust. Cloud applications must be implemented in a way that allows them to recover automatically out of the malfunctions without compromising the required Quality of Service (QoS) or the projected return.

Fault Tolerance within Cloud Computing

Cloud computing systems use a variety of fault tolerance techniques, including proactive, reactive, and task resubmission. Cloud-based applications are designed to manage failures in cloud infrastructure automatically. In order to prevent failure and increase capacity and throughput, they replace suspect components with other effective components. Reactive fault tolerance techniques can be used to increase the outcome of success on application execution, allowing applications to continue running in the event of a failure. Numerous reactive defect tolerance techniques are deployable in cloud computing systems. Checkpoints allow cloud-based systems to recover from failure and continue application execution near the point of failure. To ensure that the application state is preserved in the event of a crash, it can be saved to stable storage periodically. The application can restart at the latest checkpoint if a fault occurs in a saved state. The ability to resume execution after a fault enables the application to tolerate faults and reduces the time spent by the application in recovering from such faults.

The execution may be resumed on the similar VM or on another available VM. This approach wastes additional time. Recovery of a failing VM is required if there is only one VM available for application execution or if multiple VMs are available and can be rescheduled. However, this method is appropriate if it has single instance from the desired VM.

The replication method presupposes that the possibility of a single VM failing is substantially larger than the possibility of numerous VMs failing simultaneously. By launching multiple instances of an application on separate virtual machines (VMs), application virtualization prevents the need for recompilation. The cloud may remain delivering the services even if some instances fail because of the redundant copies. Multi-version and parallel approaches are two methods for duplicating data. In the multiversity, the application is duplicated and run on several virtual machines simultaneously. Time to results is more crucial than correctness of outcomes for parallel mechanisms [11]. The reaction time of parallel methods is significantly better than that of check-pointing and multiple version schemes. Therefore, it is a viable option for mission-critical software. When verification of findings is essential, the multiverse approach is recommended. Resubmitting tasks is the standard method of error recovery in modern scientific workflow systems. A resubmission for the resource occurs at runtime if a failed task is identified.

Background Study Cloud Architecture

Figure 1depicts a high-level overview of cloud computing architecture. The three prime objects are the Allocator, the Virtual Machines (VMs) and the Resources. The Allocator is a software component which guides the cloud services provider and customers interact. It needs the below modules to be included:

1. QoS Controller: Key role of this unit is checking the cloud computing that can satisfy the needs of customers inside a maintained QoS environment. A quality of service (QoS) controller accepts a demand from a customer accompanied by his QoS needs. This launches a query demand for appropriate virtual machines for VM database, and gets there spouse. The QoS controller will accept or reject requests according to the requirements of the request. If there are no VMs in a required QoS level, the request will be rejected. If there are VMs in a required QoS level, but not enough for all requests to be satisfied, the request will be accepted and sent to anal location process.

2. VMs Database:

Cloud virtual machines' (VMs') speed, memory size, number of CPUs, and bandwidth are only some of the performance characteristics and usage history detailed in the VM information report. The failure rate of each VM is also included. Our software engineers use a programme called VMs Monitor to keep tabs on and manage all of our virtual machines, and that programme sends regular updates to this database.

3. Broker:

The QoS Controller delivers consumer requests and QoS requirements to the broker. It determines which virtual machines can fulfil these requirements based on the current resource availability of each. So as to achieve binding processes between needs and VMs, the cloud broker must be aware of the availability and dependability of virtual machines.

This information can be obtained from the virtual machines database. In addition, the cloud broker plays a crucial role in determining prices for required services based on the cloud's pricing mechanisms. The budget issue checks the cost-of-service requests. For example, needs may increase fees based on supply convenience. Assessing is the base for checking the providing and need of cloud computing supplies the source allotment.

Figure 2 depicts the Broker's internal structure and interactions. Included are:

- VMFT Selection: This module's primary objective is to choose the appropriate fault tolerance technique for customer applications .The QoS Controller transmits an application and its QoS requirements to the VM agent, which is a software agent.
- VMs Classifier: This module identifies the virtual machines (VMs) that will execute applications for clients. The module comprises the VMC agent, a software agent. The VMFT selection module transmits the application's QoS requirements to this agent. Then, it communicates a query need to the database of virtual machines (VMs) to get the most current information for VMs which can attain the QoS needs. The categorization of virtual machines is based on both the duration of time the VM has been in use and its rate of failure.

Dispatcher: Once a customer application has been determined, the Dispatcher delivers it to one or more VMs for execution.

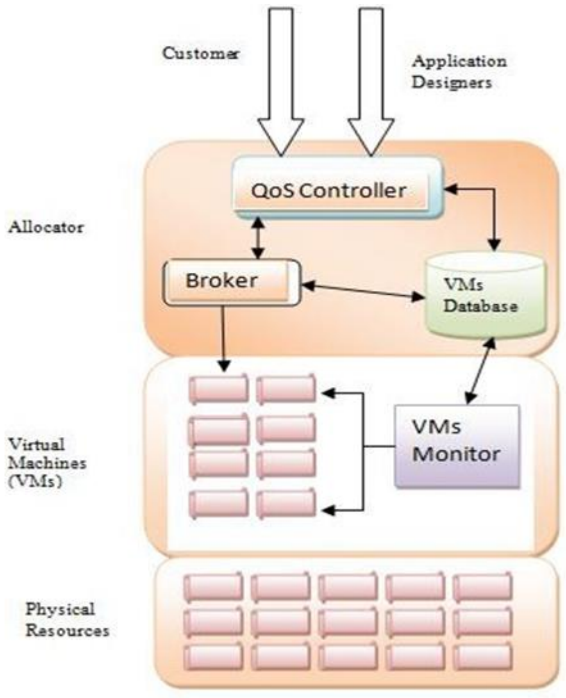


Fig 1: Overview of High level Cloud Computing *Architecture*

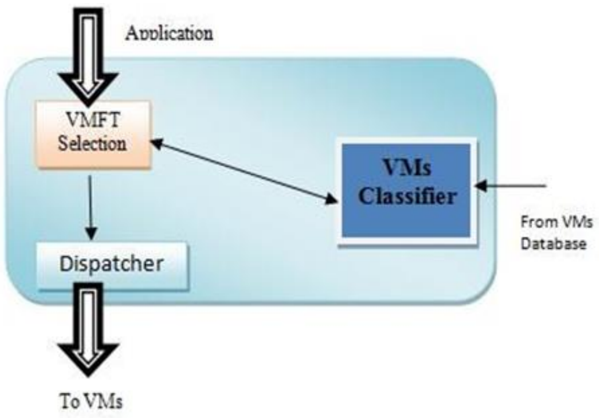


Fig 2: Broker and their interactions components

2. Related Work

Cloud computing Cloud computing workloads can be managed with the help of a fuzzy economic energy mechanism proposed by Om Kumar et al. [7]. A three-step schedule maintains cloud resources while reducing migration time and instance execution time. By deploying fuzzy decision-makers, this framework achieves workload consolidation and assesses resources with utilization concerns. Additionally, the framework supplies workload categories to monitor virtual machine failures and start fuzzy migrations. According to Amoon et al. [9], the checkpointing technique can improve cloud fault tolerance when a failure occurs. Checkpointing length is flexible in this approach. A virtual machine's failure rate can influence the length of checkpointing for an application. In order to control data center infrastructure reliability, Easwara kumar [10] introduced a new strategy called dynamic fault-tolerant VM migration (DFTM). A VN (Virtual Network) demand can be recovered using DFTM using an advanced recovery mechanism. Monitoring network traffic and load limits through VM is necessary to submit jobs to resources. Hybrid Fault-Tolerant Scheduling Algorithm Program (HFTSA) is the algorithm proposed by Yao et al. [11]. Each job is assigned a fault-tolerant cloud method from resubmissions and replications by the HFTSA scheduling algorithm. They support task characteristics and adapt cloud resources accordingly. A flexible cloud fault tolerance model (FFTF) was proposed by Hasan and Goraya [12]. An FFTF framework delivers users with a key to start fault tolerance (FT) according to their jobs. To implement FT, user jobs are done on a shared cluster in the cloud. Using support vector machines (SVMs), Beheshti and Esfahani [13] proposed a BFPF-Cloud framework to predict Byzantine failures.

There are reactive and proactive policies in the BFPF-Cloud framework for handling failures and maintaining system availability and reliability. Adaptive models developed by Alaei et al. [14] aim to reduce the total cost, energy consumption, and makespan of a system, and to tolerate faults. A reinforcement learning-based multi-work load scheduling algorithm was proposed by Zong et al [15]. A dynamic priority algorithm was utilized to check the type of progress in it. For checking the cluster nodes in cloud computing, a fine-grained cloud computing model was created using reinforcement learning. For task arrangement in virtual hosts in a cloud, Ragmani et al. [16] have proposed Fuzzy Ant Colony Improvement Algorithm Rule (FACO). For calculating the pheromone value, the fuzzy module evaluates historical data. In order to achieve the minimal computation time, a suitable server is selected. Deep Q-learning task scheduling (DQTS) is a novel computational algorithm developed by Tong et al. [17], that associates the characters of deep neural networks and Q- learning algorithms. This method is formulated for managing tasks with directed acyclic graphs (DAGs) in a cloud environment. To schedule tasks, this method uses the Deep Q-learning (DQL) technique, which primarily promotes elementary model learning to support workflow advancements. In Sahoo et al. [18], fuzzy logic was utilized for representing the various nodes in a cloud environment as a load-balancing algorithm. Choudhary and Kumar [19] propose an HG-GSA for load balancing. The firefly algorithm was created by Kashikolaei et al. [20]. A sophisticated meta-heuristic algorithm schedules and processes user requests for load balancing. Using some searching algorithms, the proposed algorithms to improve load balancing, and scheduling in cloud computing. Li et al. employed genetic and differential evolution algorithms (DEs) to minimize and increase virtual machine time, price, and load balancing [21]. Jena et al. [22] used a better Q-learning algorithm (QMPSO). Also used modified particle swarm optimization (MPSO) to dynamically balance virtual hosts. Pbest adjusts the MPSO rate over gbest to promote better Q-learning's best action. Sun et al. proposed QoS-aware scheduling [23]. With QoS-aware service in edge-cloud combined with fault-tolerance in edge- cloud, this fault-tolerant technique is created on standard primary backup (PB). The above works can be summarized as shown in Table 1.

Table1: A summary of different fault tolerance techniques

Authors	**Application**	**Techniques/ Methods**	**Findings**
Om Kumar et al.[8]	CloudSim	Reactive (Live Migration)	Adaptive fuzzy fault tolerance. Reduce execution and migration time.
Sadi et al. [9]	ACS	Reactive (Checkpoint)	Enhancement Performance is better, but the checkpoint interval isn't fixed, which makes availability low.

Sivagami et al.[10]	CloudSim	Reactive(migration)	Low resource utilization and minimal complexity. Migration time is maximum.
Zhao et al.[11]	Google Cloud trace logs	Reactive (resubmission and replication)	Response time and resource utilization are low.
Goraya et.al. [12]	CloudSim	Reactive(replication)	System scalability and resource utilization are both high. Resource consumption and response time are both low.
Esfahani et al. [13]	CloudSim	Proactive(Prediction)	Reduce the time it takes to execute and repeat. High throughput and response time are low.
Mohammed et al.[14]	CloudSim	Redundancy checkpoint	Performance is high and Utilization of resources is low.
Zohng et al.[15]	CloudSim	Workflow Scheduling	Utilization of resources is high.
Ragmaniet al, [16]	Cloud analyst	Nature Inspired	Processing time, response time and cost is maximum.
Tonget Et al. [17]	CloudSim	Workflow Scheduling	Balance of load is maximum and makespan is low.
Sahoo et al. [18]	CloudSim	Nature Inspired	Response time is high
Kumar Et al. [19]	CloudSim	Nature Inspired	Cost is low and maximum utilization
Kashikol et al.[20]	Dot Net	Task Scheduling	Improved productivity and efficiency of the resources increases due to stability

Li et al.[21]	CloudSim	Task Scheduling	Balancing of load is better. Cost and total time is minimal.
Jana et al.[22]	CloudSim	Nature Inspired	A reduction in makespan times increases system throughput and optimizes utilization of the resources.
Sun et al.[24]	Python	QoS task Scheduling	Better performance and improved service reliability.

3. Methodology

The Mann-Whitney U test is non-parametric, this is an alternative to an unpaired t-test . A null hypothesis can be tested by comparing two samples from the same population (having the same median) or by examining whether observations in one sample incline to be larger than observations in the other. Even though it is a non-parametric test, it is based on the assumption that both the distributions are analogous. The number of times a xi from sample 1 is greater than an yj from sample 2 is calculated. Ux refers to this number. Similarly, Uy indicates the number of times a xi from sample 1 is smaller than an yj from sample 2. In accordance with the null hypothesis, it is expected that Ux and Uy to be almost equal.

The test should be conducted as follows:

1. The observations should be arranged in order of magnitude.

2. To indicate which sample each observation belongs to, write X or Y (or some other appropriate symbol) under it.

3. Write down the number of ys that are to the left (smaller than it); this indicates that xi > yj. The number of xs to the left of each y indicates that yj is greater than xi.

4. Calculate the total number of times that xi > yj - denoted by Ux. The total number of times that yj> xi - denoted by Uy - must be added up. Make sure the sum of Ux and Uy equals nxny.

5. Minimum (Ux, Uy) = Calculate U

6. Calculate the possibility of accessing a value U or lower using statistical tables for the Mann-Whitney U test. For one-sided tests, this is the p-value; for two-sided tests, double this possibility to obtain the p-value.

$$\mu U = \frac{n_x\, n_y}{2}, \quad \sigma U = \sqrt{\frac{n_x\, n_y\,(N+1)}{12}}, \text{ where } N = n_x + n_y$$

nxny is large enough (> 20), Two or more observations may be identical. By providing half the tie to the X value and half the tie to the Y value, we can still calculate U. If this is the case, it is necessary to adjust the standard deviation when using the normal approximation. Therefore, we have:

$$\sigma U = \sqrt{\frac{n_x\, n_y}{N(N-1)}} \times \left[\frac{N^3\ -N}{12} - \sum_{j=1}^{g} \frac{t_j^3}{12} - \frac{t_j}{12}\right]$$

Where $N = n_x + n_y$

g = the number of groups of ties

t_j = the number of tied ranks in group j

Table 2: Parameter Table

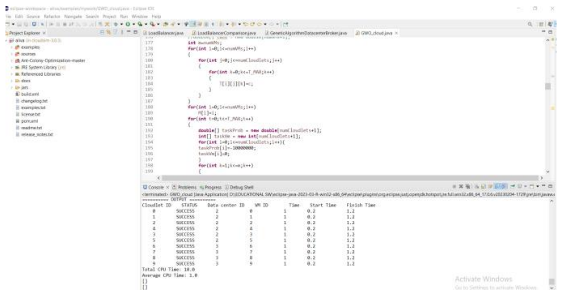
Type	Parameter	Value
DC	No. of DC	5
	No. of Hosts	2
VM	No. of VMS	20
	No. of PEs per VM	250(MIPS)
		512~2048MB

	VM Memory	
	Type of manager	Timeshared
Task	No. of task	10~50
	Length	5000MI

Algorithm

```
1. Initialize number of VMs
2. Initialize number of cloudlets
3. Initialize optimization parameters
4. For each cloudlet:
   • For each VM:
   • Calculate the time to process the task by the VM.
   • Find the VM with the minimum processing time.
5. Cluster tasks using BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies).
6. For each iteration of the optimization algorithm:
   • For each group of tasks:
   • For each cloudlet:
   • For each VM:
   • If the VM is not tabu:
   • Calculate the time to process the task by the VM.
   • Calculate the probability of processing the task by the VM.
   • If the new probability is greater than the old probability:
   ○ Update the probability.
   ○ Keep track of the VM index.
   • Update ACO parameters:
   •   Set Factor to 0.
   •   For each cloudlet:
   • Update Factor using the task probability.
   • If the new Factor is greater than the old Factor:
   • Add the VM to the VM list.
7. Calculate performance.
```

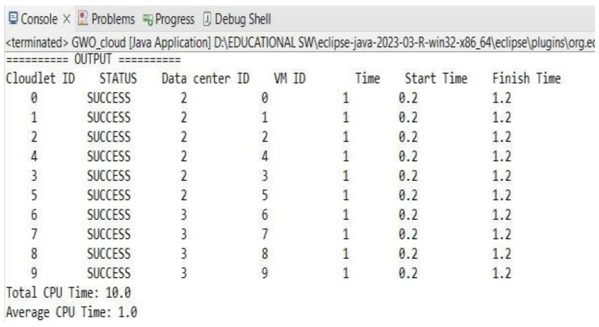
The above algorithm outlines a process for optimizing the allocation of tasks (cloudlets) to individual virtual machines (VMs) in a cloud computing environment.



```java
int m=numVMs;
for(int i=0;i<=numVMs;i++)
{
    for(int j=0;j<=numCloudlets;j++)
    {
        for(int k=0;k<=T_MAX;k++)
        {
            T[i][j][k]=c;
        }
    }
}
for(int i=0;i<=numVMs;i++)
    M[i]=i;
for(int t=0;t<=T_MAX;t++)
{
    double[] taskProb = new double[numCloudlets+1];
    int[] taskVm = new int[numCloudlets+1];
    for(int i=0;i<=numCloudlets;i++){
    taskProb[i]=-100000000;
    taskVm[i]=0;
    }
    for(int k=1;k<=m;k++)
    {
```

```
========== OUTPUT ==========
Cloudlet ID    STATUS    Data center ID    VM ID    Time    Start Time    Finish Time
    0         SUCCESS        2               0         1       0.2          1.2
    1         SUCCESS        2               1         1       0.2          1.2
    2         SUCCESS        2               2         1       0.2          1.2
    4         SUCCESS        2               4         1       0.2          1.2
    3         SUCCESS        2               3         1       0.2          1.2
    5         SUCCESS        2               5         1       0.2          1.2
    6         SUCCESS        3               6         1       0.2          1.2
    7         SUCCESS        3               7         1       0.2          1.2
    8         SUCCESS        3               8         1       0.2          1.2
    9         SUCCESS        3               9         1       0.2          1.2
Total CPU Time: 10.0
Average CPU Time: 1.0
[]
[]
```

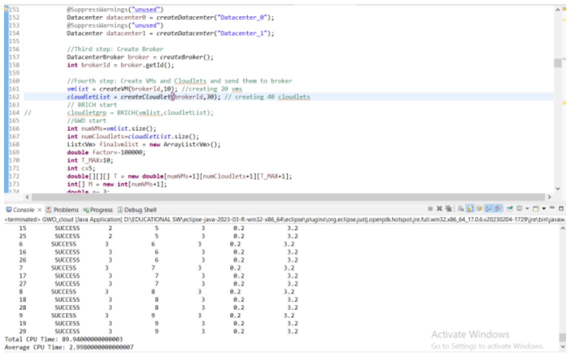
Fig 3 (a): Implementation of the algorithm for 10 tasks



```
<terminated> GWO_cloud [Java Application] D:\EDUCATIONAL SW\eclipse-java-2023-03-R-win32-x86_64\eclipse\plugins\org.ec
========== OUTPUT ==========
Cloudlet ID    STATUS    Data center ID    VM ID    Time    Start Time    Finish Time
    0         SUCCESS        2               0         1       0.2          1.2
    1         SUCCESS        2               1         1       0.2          1.2
    2         SUCCESS        2               2         1       0.2          1.2
    4         SUCCESS        2               4         1       0.2          1.2
    3         SUCCESS        2               3         1       0.2          1.2
    5         SUCCESS        2               5         1       0.2          1.2
    6         SUCCESS        3               6         1       0.2          1.2
    7         SUCCESS        3               7         1       0.2          1.2
    8         SUCCESS        3               8         1       0.2          1.2
    9         SUCCESS        3               9         1       0.2          1.2
Total CPU Time: 10.0
Average CPU Time: 1.0
```

Fig 3 (b): Simulated results for 10tasks

Figures 3 (a) & 3(b) are the screenshot of simulated results for BIRCH-GWO algorithm utilization for 10 tasks. This shows it can balance all the loads and able to tolerate the fault in a less time period successfully.



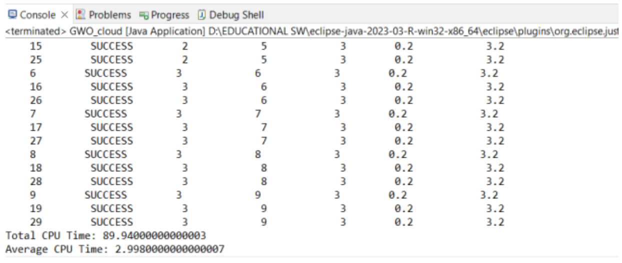
```
@SuppressWarnings("unused")
Datacenter datacenter0 = createDatacenter("Datacenter_0");
@SuppressWarnings("unused")
Datacenter datacenter1 = createDatacenter("Datacenter_1");

//Third step: Create Broker
DatacenterBroker broker = createBroker();
int brokerId = broker.getId();

//Fourth step: Create VMs and Cloudlets and send them to broker
vmlist = createVM(brokerId,10); //creating 20 vms
cloudletList = createCloudlet(brokerId,30); // creating 40 cloudlets
// BRICH start
//          cloudletgrp = BRICH(vmlist,cloudletList);
//GWO start
int numVMs=vmlist.size();
int numCloudlets=cloudletList.size();
List<Vm> finalvmlist = new ArrayList<Vm>();
double factor=-100000;
int T_MAX=10;
int c=5;
double[][][] T = new double[numVMs+1][numCloudlets+1][T_MAX+1];
int[] M = new int[numVMs+1];
```

```
<terminated> GWO_cloud [Java Application] D:\EDUCATIONAL SW\eclipse-java-2023-03-R-win32-x86_64\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.6.v20230204-1729\jre\bin\javaw.
    15       SUCCESS      2         5        3      0.2        3.2
    25       SUCCESS      2         5        3      0.2        3.2
    6       SUCCESS      3         6        3      0.2        3.2
    16       SUCCESS      3         6        3      0.2        3.2
    26       SUCCESS      3         6        3      0.2        3.2
    7       SUCCESS      3         7        3      0.2        3.2
    17       SUCCESS      3         7        3      0.2        3.2
    27       SUCCESS      3         7        3      0.2        3.2
    8       SUCCESS      3         8        3      0.2        3.2
    18       SUCCESS      3         8        3      0.2        3.2
    28       SUCCESS      3         8        3      0.2        3.2
    9       SUCCESS      3         9        3      0.2        3.2
    19       SUCCESS      3         9        3      0.2        3.2
    29       SUCCESS      3         9        3      0.2        3.2
Total CPU Time: 89.94000000000003
Average CPU Time: 2.9980000000000007
```

Fig 3 (c): Implementation of the algorithm for 30 tasks



```
<terminated> GWO_cloud [Java Application] D:\EDUCATIONAL SW\eclipse-java-2023-03-R-win32-x86_64\eclipse\plugins\org.eclipse.just
    15       SUCCESS      2         5        3      0.2        3.2
    25       SUCCESS      2         5        3      0.2        3.2
    6       SUCCESS      3         6        3      0.2        3.2
    16       SUCCESS      3         6        3      0.2        3.2
    26       SUCCESS      3         6        3      0.2        3.2
    7       SUCCESS      3         7        3      0.2        3.2
    17       SUCCESS      3         7        3      0.2        3.2
    27       SUCCESS      3         7        3      0.2        3.2
    8       SUCCESS      3         8        3      0.2        3.2
    18       SUCCESS      3         8        3      0.2        3.2
    28       SUCCESS      3         8        3      0.2        3.2
    9       SUCCESS      3         9        3      0.2        3.2
    19       SUCCESS      3         9        3      0.2        3.2
    29       SUCCESS      3         9        3      0.2        3.2
Total CPU Time: 89.94000000000003
Average CPU Time: 2.9980000000000007
```

Fig 3 (d): Simulated results for 30tasks

Figures 3 (c) & 3(d) are the screenshot of simulated results for BIRCH-GWO algorithm utilization for 30 tasks. This shows the algorithm can successfully balance all the loads and tolerate the fault with a minimum CPU time.

4. Result & Discussion

Above figure shows with varying number of tasks how makespan time of different models get affected. The degree of imbalance is better with more number of tasks while using optimization algorithms. As because with less number of tasks the objective of finding a suitable VM is not that much necessary and each task get processed by any VM. Above figure taken from figure-2 to give clarity on how makespan is reduced with application of optimization.

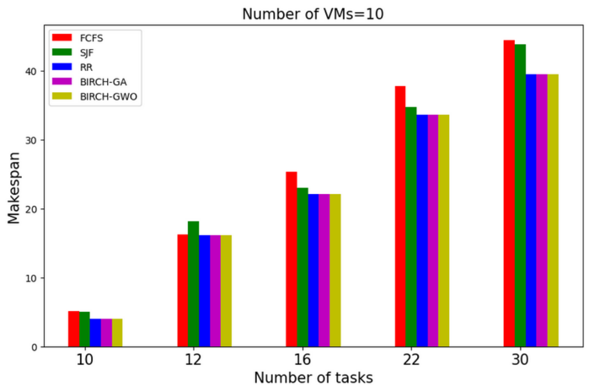


Fig-4: simulated graph with 10 VMs

Figure 4 shows with varying numbers of tasks how the makespan time of different models get affected. We have utilized 5 different algorithms on the task variation from 10 to 30. We have varied the number of tasks, but the number of VMs for all the algorithms remains the same as 10. We observed that the variation with different algorithms would be minimal with fewer tasks, but when no. of tasks increased, the makespan time gets affected.

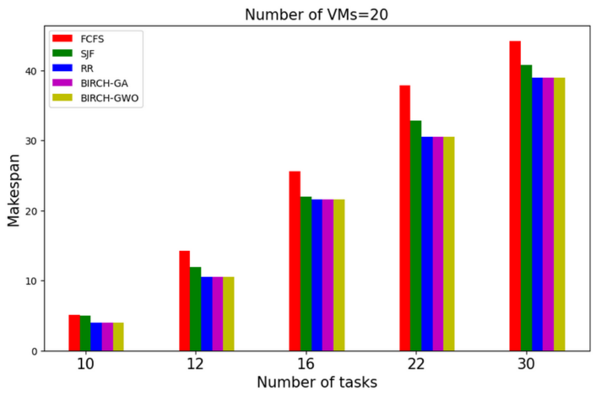


Fig-5: simulated graph with 20 VMs

Figure 5 shows with varying numbers of tasks how the makespan time of different models get affected. We have utilized 5 different algorithms on the task variation from 10 to 30. We have varied the number of tasks, but the number of VMs for all the algorithms remains the same at 20. We observed that the variation with different algorithms would be minimal with fewer tasks, but when no. of tasks increased, the makespan time gets affected.

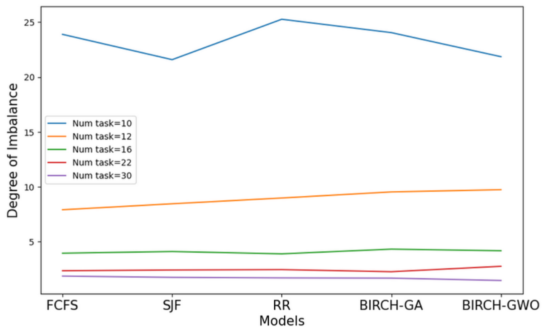


Fig-6: simulated graph with different no. of tasks

In Figure 6, we observed that the degree of imbalance is better with more tasks while using optimization algorithms. As with fewer tasks, the objective of finding a suitable VM is not that much required, and each task gets processed by any VM, so more tasks can provide better clarity on utilizing proper algorithms.

In Figure 7, we tried to clarify how makespan timings are reduced with 10 no. of tasks by utilizing all 5 different algorithms for optimization application.

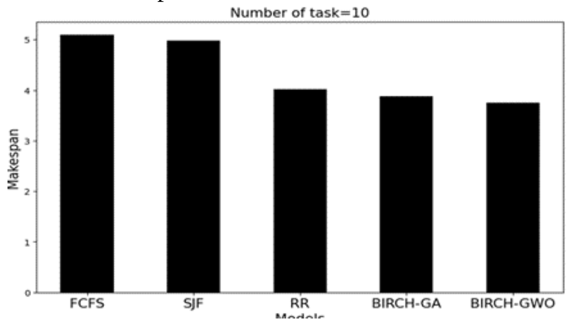


Fig-7: Simulated graph with 10 tasks

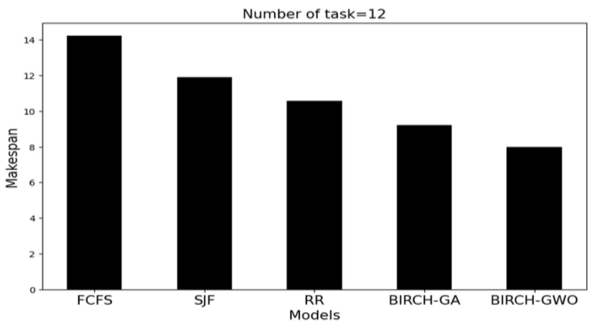


Fig-8: simulated graph with 12 tasks

In Figure 8, we tried to clarify how makespan timings are reduced with 12 no. of tasks by utilizing all 5 different algorithms for optimization application.

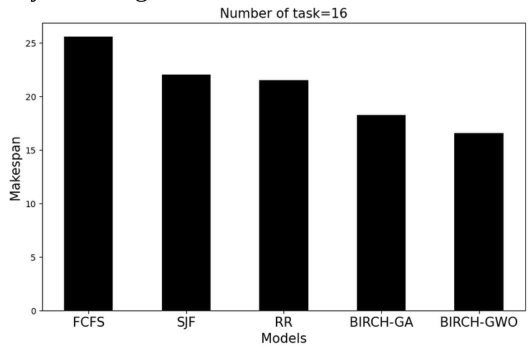


Fig 9: simulated graph with 16 tasks

In this figure 9, we tried to clarify how makespan timings are reduced with 16 no. of tasks by utilizing all 5 different algorithms for optimization application. Figure 10 shows our best efforts to explain how combining all 5 optimization strategies can shorten makespan times for 22 tasks.

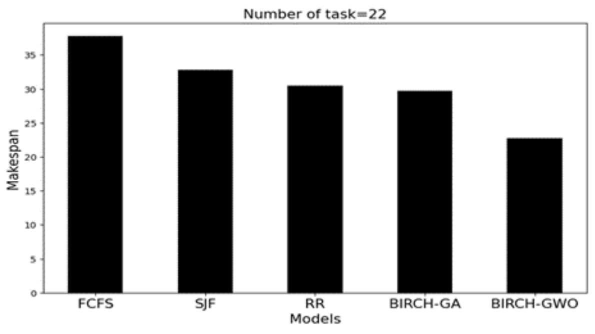


Fig-10: simulated graph with 22 tasks

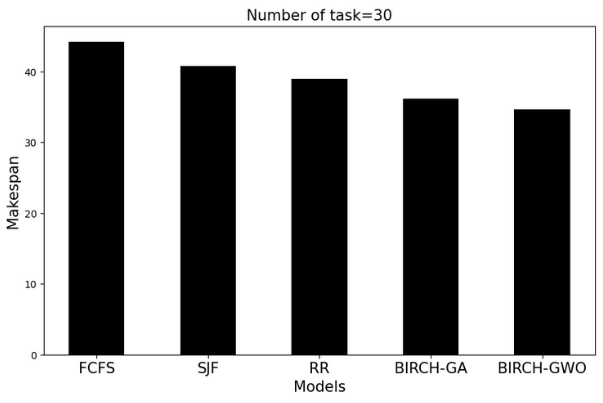


Fig-11: simulated graph with 30 tasks

We have clarified how makespan timings are reduced with 30 no. of tasks by utilizing all 5 different algorithms for optimization application shown in Figure 11.

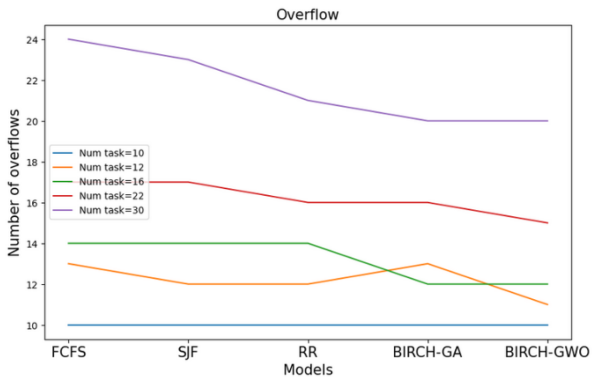


Fig 12: simulated graph for checking overflow

Overflow occurrences are depicted graphically in Figure 12. When more jobs are made available for load balancing, BIRCH-GWO can reduce the occurrence of overflows. Number of time overflow occurred has been shown above. With application of BIRCH-GWO numbers of overflows are reduced specifically for more number of tasks.

5. Conclusion

This paper introduces a dynamic fault tolerance management algorithm for VM migration in cloud data centers. The algorithm is based on a layered modeling architecture that combines reactive and proactive fault modeling theories to ensure reliable and fault-tolerant cloud computing systems. The algorithm minimizes service interruptions and maximizes resource utilization by considering defect prediction, and resource allocation techniques. The algorithm's effectiveness in reducing service downtime, ensuring application reliability, and sustaining optimal performance has been demonstrated through extensive simulations and evaluations. The dynamic VM migration capability based on defect prediction enhances resource allocation and load balancing, contributing to system resilience. The proposed algorithm provides a comprehensive approach to maintaining fault tolerance and recovery in cloud data centers, making cloud-based applications more dependable. By resolving fault tolerance concerns through a layered modeling approach, the algorithm offers a robust and reliable solution for managing fault tolerance in cloud computing environments. This research contributes to developing fault-tolerant cloud systems by presenting a practical algorithm for dynamic fault tolerance management and VM migration in cloud data centers.

Overall, this paper highlights the effectiveness of the *BIRCH-GWO* algorithm in reducing overflows and emphasizes the importance of considering the number of tasks when optimizing task scheduling algorithms in cloud environments. The findings suggest potential avenues for future research to enhance the dynamic modification of task numbers and further optimize task allocation and scheduling algorithms in cloud computing.

Acknowledgements

This research was partially supported by Dr. Saumendra Pattnaik and Prof. (Dr.) Binod Kumar Pattanayak. We thank our colleagues from Siksha 'O' Anusandhan University, Bhubaneswar, Odisha, India, who provided insight and expertise that greatly assisted the research.

Author contributions

Bikash Chandra1 Pattanaik1: Related works, Experimental analysis & Implementation, Abstract **Bidush Kumar2 Sahoo2:** Result & Discussion, Abstract **Bibudhendu3 Pati3**: Methodology, References, Abstract **Suprava Ranjan4 Laha4**: Introduction, Background Study, Conclusion, Abstract.

Conflicts of interest

The authors declare no conflicts of interest.

References

[1] Gowri, A. S. (2019). Impact of virtualization technologies in the development and management of cloud applications. *International Journal of Intelligent Systems and Applications in Engineering*, *7*(2), 104-110.

[2] Ahmet, E. F. E., & Isık, A. (2020). A general view of industry 4.0 revolution from cybersecurity perspective. *International Journal of Intelligent Systems and Applications in Engineering*, *8*(1), 11-20.

[3] Pattanayak, B. K., Mohanty, R. K., Venkataramana, T., Pattnaik, S., Ranjan, R., Laha, S. R. (2022) An IoT Enabled Cloud Based Virtual Classroom Assisted e-Learning System. In NeuroQuantology, Vol.20, 9.

[4] Laha, S. R., Parhi, M., Pattnaik, S., Pattanayak, B. K., & Patnaik, S. (2020, October). Issues, Challenges and Techniques for Resource Provisioning in Computing Environment. In *2020 2nd International Conference on Applied Machine Learning (ICAML)* (pp. 157-161). IEEE.

[5] Hota, N., & Pattanayak, B. K. (2021). Cloud computing load balancing using Amazon web service technology. In *Progress in Advanced Computing and Intelligent Engineering* (pp. 661-669). Springer, Singapore.

[6] Laha, S. R., Mahapatra, S. K., Pattnaik, S., Pattanayak, B. K., & Pati, B. (2021). U-INS: an android-based navigation system. In *Cognitive Informatics and Soft Computing* (pp. 125-132). Springer, Singapore.

[7] The worst cloud outages of 2013, Online, cited 01.07.2013.URL:http://www.infoworld.com/slideshow/107783/the-worst-cloud-outages of 2013-so-far-221831.

[8] The worst cloud outages of 2014, Online, cited 01.01.2015.URL:http://www.infoworld.com/article/2606209/cloud-computing/162288-The-worst-cloud-outages-of-2014-so-far.html.

[9] CU, O. K., & Bhama, P. R. S. (2019). Fuzzy based energy efficient workload management system for flash crowd. *Computer Communications*, *147*, 225-234.

[10] Amoon, M., El-Bahnasawy, N., Sadi, S., & Wagdi, M. (2019). On the design of reactive approach with flexible checkpoint interval to tolerate faults in cloud computing systems. *Journal of Ambient Intelligence and Humanized Computing*, *10*(11), 4567-4577.

[11] Sivagami, V. M., & Easwarakumar, K. S. (2019). An improved dynamic fault tolerant management algorithm during VM migration in cloud data center. *Future Generation Computer Systems*, *98*, 35-43.

[12] Yao, G., Ren, Q., Li, X., Zhao, S., & Ruiz, R. (2020). A hybrid fault-tolerant scheduling for deadline-constrained tasks in Cloud systems. *IEEE Transactions on Services Computing*.

[13] Hasan, M., & Goraya, M. S. (2019). Flexible fault tolerance in cloud through replicated cooperative resource group. *Computer Communications*, *145*, 176-192.

[14] Beheshti, M. K., & Safi-Esfahani, F. (2020). BFPF-Cloud: Applying SVM for Byzantine Failure Prediction to Increase Availability and Failure Tolerance in Cloud Computing. *SN Computer Science*, *1*(5), 1-31.

[15] Alaei, M., Khorsand, R., & Ramezanpour, M. (2021). An adaptive fault detector strategy for scientific workflow scheduling based on improved differential evolution algorithm in cloud. *Applied Soft Computing*, *99*, 106895.

[16] Zhong, J. H., Peng, Z. P., Li, Q. R., & He, J. G. (2019). Multi workflow fair scheduling scheme research based on reinforcement learning. *Procedia Computer Science*, *154*, 117-123.

[17] Ragmani, A., Elomri, A., Abghour, N., Moussaid, K., & Rida, M. (2020). FACO: A hybrid fuzzy ant colony optimization algorithm for virtual machine scheduling in high-performance cloud computing. *Journal of Ambient Intelligence and Humanized Computing*, *11*(10), 3975-3987.

[18] Tong, Z., Chen, H., Deng, X., Li, K., & Li, K. (2020). A scheduling scheme in the cloud computing environment using deep Q-learning. *Information Sciences*, *512*, 1170-1191.

[19] Hamdani, M., Aklouf, Y., & Bouarara, H. A. (2019, March). Improved fuzzy load-balancing algorithm for cloud computing system. In *Proceedings of the 9th International Conference on Information Systems and Technologies* (pp. 1-4).

[20] Chaudhary, D., & Kumar, B. (2019). Cost optimized hybrid genetic-gravitational search algorithm for load scheduling in cloud computing. *Applied Soft Computing*, *83*, 105627.

[21] Kashikolaei, S. M. G., Hosseinabadi, A. A. R., Saemi, B., Shareh, M. B., Sangaiah, A. K., & Bian, G. B. (2020). An enhancement of task scheduling in cloud computing based on imperialist competitive algorithm and firefly algorithm. *The Journal of Supercomputing*, *76*(8), 6302-6329.

[22] Li, Y., Wang, S., Hong, X., & Li, Y. (2018, July). Multi-objective task scheduling optimization in cloud computing based on genetic algorithm and differential evolution algorithm. In *2018 37th Chinese Control Conference (CCC)* (pp. 4489-4494). IEEE.

[23] Jena, U. K., Das, P. K., & Kabat, M. R. (2020). Hybridization of meta-heuristic algorithm for load balancing in cloud computing environment. *Journal of King Saud University-Computer and Information Sciences*.

[24] Sun, H., Yu, H., Fan, G., & Chen, L. (2020). QoS-aware task placement with fault-tolerance in the edge-cloud. *IEEE Access*, *8*, 77987-78003.

[25] Parshapa, P. ., & Rani, P. I. . (2023). A Survey on an Effective Identification and Analysis for Brain Tumour Diagnosis using Machine Learning Technique. International Journal on Recent and Innovation Trends in Computing and Communication, 11(3), 68–78. https://doi.org/10.17762/ijritcc.v11i3.6203

[26] Sahoo, D. K. . (2022). A Novel Method to Improve the Detection of Glaucoma Disease Using Machine Learning. Research Journal of Computer Systems and Engineering, 3(1), 67–72. Retrieved from https://technicaljournals.org/RJCSE/index.php/journal/article/view/44