World Journal of Advanced Research and Reviews
eISSN: 2581-9615 CODEN (USA): WJARAI
Cross Ref DOI: 10.30574/wjarr
Journal homepage: https://wjarr.com/

(RESEARCH ARTICLE)

Check for updates

Development of a fault-tolerant management scheme for on demand real time data centre cloud environment

SANUSI L. A [1,*] ADEOSUN O.O [2], AFOLABI A.O [2], ISMAILA W.O [2] and SANUSI H.A [3]

[1] *Postgraduate Researcher, Computer Science and Informatics Lautech, Ogbomoso, Nigeria.*
[2] *Lecturer, Department of Computer Science and Informatics Lautech, Ogbomoso, Nigeria.*
[3] *Department of Computer Science and Engineering Lautech, Ogbomoso, Nigeria.*

World Journal of Advanced Research and Reviews, 2024, 24(03), 1554–1566

Publication history: Received on 27 August 2024; revised on 14 December 2024; accepted on 16 December 2024

Article DOI: https://doi.org/10.30574/wjarr.2024.24.3.3072

Abstract

Fault tolerance in cloud data centers is a critical mechanism for handling the escalating frequency of failures. As the size and complexity of large-scale systems grow, the challenge of predicting and mitigating failures becomes increasingly exponential, rendering previous solutions inadequate for meeting the high-performance demands of both cloud users and providers. This paper aims to address the growing need for improved fault tolerance by developing a management computational intelligence scheme tailored for real-time, on-demand cloud data center environments. In this study, a fault-tolerant computational intelligence scheme was elicited for the parameters necessary for the design. A computational system was also designed to determine league winners based on scheduling checkpoints. Also, the designed system was simulated using java programming language in CloudSim simulation toolkit (3.0.3) with a customized Cloud Analyst Graphic User Interface (GUI) on the Eclipse Integrated Development Environment (IDE) Luna release 4.4. The developed system (Checkpointed League Winner Algorithm) was compared with existing scheme such as Ant Colony Optimization (ACO), Genetic Algorithm (GA) and League Championship Algorithm (LCA) in real time. Checkpointed League Winner Algorithm was also evaluated to check the scheme's resistance to faults and the improvement percentage of the cloud data centres. The parameters used to evaluate the scheme are: failure to perform Ratio (FPR), Failure that causes a Delay in Performance (FDP), and the Rate at which Performance improves (RPI). The result indicates that when the whole average life of each scheme is considered, Checkpointed League Winner Algorithm (CPLWA) results in a 38.2%, 29.9%, and 20.5% improvement over ACO, GA, and LCA, respectively. The average makespan of the scheme indicates that the Checkpointed League Winner Algorithm exhibits a significant improvement, outperforming the ACO, GA, and LCA with 41%, 33%, and 23%, respectively; the response time of the scheme indicates that the Checkpointed League Winner Algorithm outperformed the ACO, GA, and LCA with 54.3%, 56.6%, and 30.2%, respectively; and the failure ratio of the scheme indicates that the Checkpointed League Winner Algorithm performs better than existing meta-heuristics methods (ACO, GA, and LCA) with a lower failure ratio. This improvement can be attributed to the iterative structure, the migration, and the checkpointing approaches employed in the scheme. This study developed a fault tolerance computational intelligence scheme for an on-demand real-time data centre cloud environment in which the Checkpointed League Winner Algorithm outperformed the existing scheme in terms of response time, average makepan and failure ratio. It is then suggested to explore the application of the Checkpointed League Winner Algorithm scheme to address resource management, provisioning, and virtual machine placement challenges within distributed systems.

Keywords: Fault Tolerance; Cloud Computing; System Failure; Cloud Data Centers; Cloud Sim

* Corresponding author: SANUSI L. A.

1. Introduction

The rapid expansion and increasing complexity of large-scale systems have brought about significant challenges in predicting failures and mitigating the consequences of system breakdowns. Traditional failure management techniques often fall short in meeting the high-performance requirements demanded by both cloud users and providers (Mohammed et al., 2019). Cloud infrastructures, particularly under heavy use, are susceptible to failures at the cloud node level, which can disrupt service availability. These failures can stem from various causes, including software bugs, hardware malfunctions, human errors, and malicious attacks (Kumari, 2018). Despite advancements in hardware functionality within cloud infrastructures, failure rates remain notably high (Dwivedi, 2018).

Cloud computing has become a focal point in both academic research and industrial applications due to its on-demand flexibility, scalability, and cost-efficiency, eliminating the need for significant upfront investments (Sefraoui et al., 2014). According to the National Institute of Standards and Technology (NIST), cloud computing is defined as a model that enables convenient, on-demand network access to a shared pool of configurable computing resources, such as networks, storage, servers, applications, and services, which can be quickly provisioned and released with minimal management effort (Mell & Grance, 2011). This technology encompasses a wide range of technologies, including virtualization, distributed computing, utility computing, and emerging trends in networking and software services. Cloud computing relies on hardware and software managed by third-party providers at remote locations, allowing individuals and organizations to access various services, such as social networking, online file storage, webmail, and business applications. These applications often require robust fault tolerance and high dependability to ensure uninterrupted operation.

Virtualization technologies play a crucial role in cloud environments by consolidating and managing resources effectively. The growing demand for computational capacity and resources driven by developments in artificial intelligence, machine learning, big data, and cloud-based high-performance systems has heightened the potential for system failures (Gainaru & Cappello, 2015). Implementing a reliable fault tolerance management system is vital for the seamless operation of data center infrastructures, ensuring high availability and dependability even in the face of unexpected failures (Patidar et al., 2011). High-performance computing (HPC) systems, often relied upon by scientists and researchers for complex computational tasks, underscore the critical importance of fault tolerance (Egwutuoha et al., 2012). Outsourcing IT services to external providers, as seen in the cloud computing model, offers the advantage of paying only for utilized resources, with cloud providers managing the underlying infrastructure. However, this model also increases the risk of node failures, which can lead to frequent interruptions in long-running applications, necessitating a rollback to previous states.

Real-time systems, which must produce responses within stringent time frames, depend on both timely delivery and logical accuracy to maintain dependability (Stankovic, 1988; Unsal et al., 2002). The challenge of enhancing fault tolerance in cloud data centers involves implementing mechanisms to effectively manage failures, given the increasing frequency of such occurrences. Ensuring system resilience and dependability is paramount in this context (Hosseini et al., 2015). The study aims to develop a fault-tolerant management scheme for on-demand real-time data center cloud environments. This involves designing a robust framework, developing the system, simulating it using the CloudSim simulation toolkit, and evaluating its performance based on response time and average makespan. The significance of this research lies in its potential to ensure continuous service availability, reduce downtime costs, and enhance the reliability of cloud services, ultimately contributing to advancements in cloud computing and inspiring further research in fault tolerance management.

2. Related studies

Kaur and Kinger (2014) investigated the integration of fault tolerance within advanced cloud computing environments, focusing on First Come First Serve (FCFS) and shortest job first principles. Their study also analyzed misses per instruction in extensive fault tolerance methods. The proposed algorithm aimed to enhance reactive fault tolerance by reallocating tasks from faulty servers to new servers with minimal load during fault occurrences within the cloud infrastructure.

Oikonomou et al. (2015) explored efficient virtual machine provisioning in cloud data centers using CloudSim as a simulation platform. Their mechanism aimed to reduce the number of active hosts, decrease energy consumption, and

minimize virtual machine migrations and service level agreement violations. However, their study did not consider failure incidents within virtual machines in cloud infrastructure.

Qasem and Madhu (2017) examined proactive fault tolerance in cloud data centers for performance efficiency. They proposed a Fuzzy Min-Max Neural Network classification approach to predict faults, anticipating failures in advance. Using CloudSim, their study demonstrated the practical possibility of predicting virtual machine failures, showing that their approach yielded better prediction results for cloud data centers.

Shwe and Aritsugi (2017) introduced a proactive replication strategy using CloudSim, incorporating predicted CPU and disk utilization, and the popularity of replicas to manage replication while balancing server workloads. Their approach considered the reliability of data blocks and node performance status for re-replication decisions. However, it did not apply high-accuracy prediction methods.

Meroufel and Belalem (2017) proposed a smart checkpoint infrastructure for cloud computing tasks, storing checkpoints in pre-paid alternative virtual machines to expedite task resumption and reduce costs following node crashes. Their study demonstrated this strategy's effectiveness in terms of energy consumption, system-level aggregation, violation mitigation, and reliability.

Kurziborae et al. (2019) developed methods and tools for identifying abnormal situations during large-scale computational experiments in high-performance computing environments. Their approach focused on localizing malfunctions, automatically troubleshooting, and reconfiguring the computing environment. By integrating monitoring systems from different nodes into a unified meta-monitoring system, they minimized diagnostic and troubleshooting time through parallel operations, enhancing computing environment resiliency.

Ragmani et al. (2020) examined an adaptive fault tolerance model to improve cloud computing performance using an artificial neural network. Their study explored the correlation between hard drive parameters and failure, leveraging attributes of self-monitoring, analysis, and reporting technology (SMART). They also evaluated the predictive capabilities of five machine learning models, including naïve Bayes and artificial neural network (ANN), to define a failure prediction module for cloud architecture. Their experimental results indicated that the artificial neural network model offered the best prediction accuracy.

The increasing reliance on cloud computing across various industries necessitates robust and fault-tolerant management schemes in data center environments. This research aims to develop a fault-tolerant management scheme for on-demand real-time data center cloud environments, focusing on identifying and mitigating potential failures to ensure continuous service availability. The proposed scheme will be evaluated through simulations and real-world case studies to demonstrate its effectiveness in enhancing the reliability and resilience of data center clouds. The ultimate goal is to provide a reliable and cost-effective solution for managing data center clouds in on-demand real-time scenarios.

3. Methodology

3.1. Research Approach

The developed management framework for cloud data centres in this research is presented in Figure 1. The following methodologies was carried out in order to achieve the stated aim for this research.

- For the parameters required for the design that influence the frequent attribute of the scheduling system in large-scale distributed environments and scheduling issues in the cloud environment of data centers, a fault tolerance computational intelligence scheme is elicited.
- A fault tolerance computational system is designed based on using an algorithm on task leagues winner based on scheduling checkpoints.
- The developed system is simulated (i.e the developed scheme) and compared with the existing scheme such as Ant Colony Optimization and Genetic Algorithm, in real time using java programming language in Cloudsim simulation toolkit (3.0.3) with a customized Cloud Analyst GUI interface on the Eclipse IDE luna release 4.4 was used to simulate the method performance.
- The system was evaluated in other to check the scheme resistant to faults and the improvement percentage of the cloud data centres. The following parameters were used to evaluate the scheme such as Failure to Perform Ratio (FPR), Failure that causes a Delay in Performance (FDP), Rate at which Performance Improve (RPI).

3.2. Scheduling Strategies Design of Fault Tolerance Data Centres Framework

This section describes in depth the functionality and harmonic functions of each of the components that combine to form the entire system.

3.2.1. Check-pointing and Restarting the Data Centers

This component is going to periodically record the task's state as it is being executed. In the event that the job fails, it resumes the failed task on the same virtual machine, starting from the point of failure. In the event that this is not the case, the operation is resumed from the most recent known state and the computer is moved to another virtual machine. In the event of a task or virtual machine (VM) failure, this effective fault-tolerant component regularly snaps to preserve redundant process execution in the memory. Since it enables the storing of incomplete computations in the event of a process failure and the customer is not charged for this, it is very helpful in spot occurrences. Checkpointing policies are taken at a rate set by the customer and applied as needed. On the one hand, checkpointing overhead situations are considered. The number of checkpoints needed for the duration of the failure-free interval between consecutive checkpoints must be determined first, and then the overhead of each checkpoint (γ) must be calculated. Equation (3.1) states that given a discrete checkpointing series of points (t_1, t_2,... t_n) and a continuous checkpointing incidence function (n(t), the integral of the incidence function with respect to the series of checking points equals one.

$$\int_{t_{i-1}}^{t_i} n(t)dt = 1, \ \forall i = 1 \qquad (3.1)$$

When t_i is the sequence of checkpoints in the data cloud's virtual machines (VMs) and i = 1, 2, 3, are the placements. The following equation for the checkpointing overhead can then be written as equation (3.2), given that N, T_i indicates the number of checkpoints that occurred during the period zero to T_i and is represented by the integral of the checkpoint incidence function n(t) during the interval [0, T_i].

$$\gamma . N_0, T_i = \gamma \int_0^{T_i} n(t)dt \qquad (3.2)$$

Before going into the checkpointing technique, it's important to define the terms stage, time step index t_n, and expendability of a checkpoint. A maximum number of checkpoints, represented by the letter s, is assigned and maintained by the procedure. When tagging the intermediate solutions of an original model alongside the adjoint model at a particular time, the time step index is frequently utilized. It begins at 0 for the original model's first state, increases to 1 for the solution found after the first t_n, and keeps going up as the complexity of the original model increases. When dealing with adjoint solutions, on the other hand, the t_n index declines with time until it approaches zero for the last t_n of the adjoint time inclusion. The checkpoint with t_n index i or more consistently, the checkpoint at time step i is characterized as the checkpoint that saves the intermediary solution with t_n index or the checkpoint that saves the intermediate solution (i.e. a fault) with t_n index i. As a result, if a checkpoint's t_n index is lower than that of another stated checkpoint at a higher stage, it is deemed disposable.

3.2.2. Data Center Job Transfer

All stopped jobs (Jn) are reassigned via the job migration technique to the next available under-loaded virtual machine job (VMj), also known as the backup cloud. A specific machine may not complete process implementation for some assigned jobs (Jn) due to unforeseen circumstances (e.g., task overloading or VMi breakdown). In this scenario, the stopped or terminated jobs (Jn) will be moved right away to the following virtual machine that becomes available so that the process can start right away.

3.3. Development of a Fault Tolerance Mechanism for Data Cloud Centre Environment

The League Championship Algorithm (LCA) optimization technique has been effectively applied to a wide variety of non-deterministic polynomial time problems that are both discrete and continuous in nature. It is a quick, metaheuristic optimization technique that may be used to find optimality on a local or global search space. It also has the ability to break free from a local minimum in the search space. Its recent applications in a range of research domains have shown that it can eventually match other cutting-edge methods in terms of performance. Because the LCA optimization technique effectively achieves an appropriate balance between global and local optima while maintaining scalability, its application to tackle fault tolerance in the cloud scheduling problem is therefore justified. By utilizing checkpoints in the LCA optimization technique, the study's main objective to develop a secure fault tolerance aware task scheduling system for cloud technology is achieved.

3.3.1. Model Formulation

In order to get an optimal solution to this non-deterministic polynomial time-hard scheduling issue, the problem is formalized in order to mathematically characterize the cloud data center system. In the model formulation of the system, we begin by examining a collection of separate jobs $j_1, j_2, \ldots j_n$ that must be scheduled on a succession of diverse and changing resources $r_1, r_2, \ldots r_m$ in order for the system to be executed successfully. $R = \{r_k | m \geq k \geq 1\}$is the number of resources in the group, and m is the number of resources available, j_1 is the number of jobs in the group, and m is the number of tasks that can be done in one go, $J = \{j_i | n \geq i \geq 1\}$ is a group of jobs, and the value of i is the counter for the number of jobs (Aaron et al, 2015). We considered the fact that consumers want to reduce the cost of making virtual resources available by lowering the make span, while cloud administrators want to maximize revenues in order to model the objective function. As a result, we may compute the Fitness value by employing the make span as shown in equation (3.3).

$$f(x_i^t) = \theta cost + \delta makespan \quad (3.3)$$

Where the values of θ and δ, which are masses used to calculate the fitness function's mechanism, are in the ranges of $1 > \theta \geq 0$ and $1 > \delta \geq 0$, respectively, and $f(x_i^t)$is the fitness function.

, and

$$cost = min\big(c(r_k, j_i)\big); form \geq k \geq 1, n \geq i \geq 1 \quad (3.4)$$

The fitness function is represented by the cost $c(r_k, j_i)$, which reflects the cost of tasks j_i that are done on resource r_k. Equations (3.5) to (3.8) are used to compute the cost of security assurance associated with the function.

$$c(r_k, j_i) = c_e(r_k, j_i) + c_s(r_k, j_i) \quad (3.5)$$

and where,

$$c_e(r_k, j_i) = \sum_{j_i \in J} \frac{C(j_i, r_k)}{C_{m(j_i)}} XJ \quad (3.6)$$

The completion time is defined as

$$C_{m(j_i)} = max_{j_i} \in J, r_k \in R. C(r_k, j_i) \quad (3.7)$$

Where c_e is the execution cost, c_s is the security cost, R is the resources, C is the completion time, and $J$is the entire task. Then the cost security assurance function is expressed as shown in equation (3.8).

$$c_s(r_k, j_i) = \sum_{j_i \in J} P_f(j_i, r_k) X \frac{E(j_i, r_k)}{E_{m(j_i)}} XJ \quad (3.8)$$

where E is the approximate amount of time needed to finish the relevant jobs or tasks. The execution time is the amount of time needed to complete each task on a given resource, as calculated by the E matrix. The components of the matrix are represented as $E(j_i, r_k)$, and the dimension of the matrix is obtained by dividing the number of cloud jobs by the number of resources. Furthermore, $P_f(j_i, r_k)$, is defined as the failure probability of tasks that are executed with checkpointing (γ) and a trust level, respectively (τ). As a result, the probability of executing job will be as shown in equation (3.9).

$$P_f(j_i, r_k) = \begin{cases} 0, if \gamma_i \leq \tau_k \\ 1 - e^{-\alpha(\gamma_i - \tau_k)}, if \gamma_i > \tau_k \end{cases} \quad (3.9)$$

In this case,

$$E_{m(j_i)} = max_{j_i} \in J, r_k \in R. E(j_i, r_k) \quad (3.10)$$

and

$$makespan = min(F_{j_i})j_i \in J \quad (3.11)$$

3.3.2. Mechanism for Fault Tolerance and Assumption

Fault tolerance is a prerequisite for real-time systems in the cloud because of the potentially catastrophic effects of failures occurring in both heterogeneous and homogeneous computer environments. The following elements make up the fault tolerance technique covered in this research paper: a job migration and replication mechanism, a checkpointing and restart system, and a defect detector. The application of this notion requires the following presumptions.

The datacenter is expected to generate enough cloud resources to prevent virtual machine (VM) rejections due to resource contention. The assumption isn't unduly ambitious because the datacenter can handle many fewer resources than what is needed

At any one time, only one host or virtual machine can die, resulting in the transfer of all active jobs. It can also be completed or performed by its backups before another server goes down for the count.

Faults can be either transient or stable, and they can also self-regulate by interacting with one another.

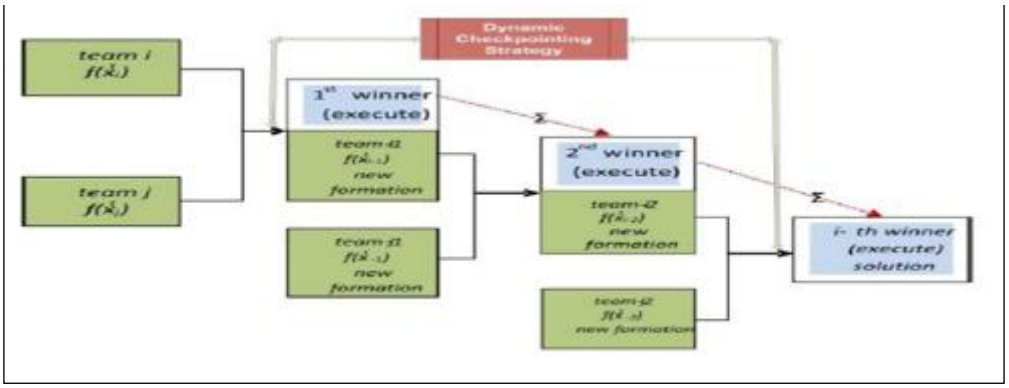

Figure 1 The CPLWA System Control Framework

3.4. Simulation Setup for the System in CloudSim

We used a cloud framework named CloudSim 3.0.3 toolkit with a modified Cloud Analyst GUI interface on the Eclipse IDE Luna release 4.4 to simulate the performance of our recommended CPLWA algorithm for job scheduling scheme (Ahmad and Khan, 2019). The job traces of the cloud data centers from Hazlewood were created in approximately 73.496 traces (Feitelsonet al., 2014). The workload archive is made publicly available through the San Diego Supercomputer Center (SDSC), and CloudSim is compatible with the Standard Workload Format (SWF) that it uses. The values of pc = 0.01 and φ_1 = φ_2 = 0.5 are the parameters for the LCA and CPLWA. The number of ants, the evaporation factor, the pheromone tracking weight, the heuristic information weight, and the pheromone updating constant Q were all adjusted to be 10, 0.01, 3, 3, and 10 respectively. These ant colony optimization (ACO) parameters were found using (Hu and Luo, 2018). Additionally, the genetic algorithm (GA) values were ascertained by setting the crossover rate to 0.5, the mutation rate to 0.1, the number of population sizes to 1000, and the number of maximum iterations to 1000 using the framework in Gatsior and Seredyński, 2013. Four datacenters were added to the system, and twenty cloud users were created, each with twenty associated cloud brokers. There are five servers in total in each of the datacenters. The Xen hypervisor, which runs on Linux, kept an eye on eight virtual machines (VMs) with 512 MB of RAM, a 10000 MB image size, and a single processor. All of these VMs were used with the Time-Shared policy. Following a 24-hour operation, the data center system is assigned 2048 Mb of host memory, 1000000 Mb of storage capacity, and 10000 Mb/s of bandwidth. Additionally, job traces ranging in length from 100 to 1000 minutes and in file size from 600 to 800000 bytes were made available for comparison. The metaheuristics ACO, LCA, and GA were employed in the comparison process. Up to twenty simulations of each algorithm were required, with the result being a distinct mean makespan, failure rate, and RPI value after each simulation.

4. Results and discussion

4.1. Measuring the Effectiveness of the System Framework

The effectiveness of the developed Checkpoint League Winner Algorithm (CPLWA), fault-tolerant management scheme is evaluated by measuring several parameters related to fault tolerance. These parameters include the Failure to Perform Ratio (FPR), Failure that Causes a Delay in Performance (FDP), and the Rate at which Performance Improves (RPI). The Failure to Perform Ratio (FPR) metric is used to assess the efficiency of the CPLWA fault tolerance scheme to indicate when a task or scheduled job fails to perform.

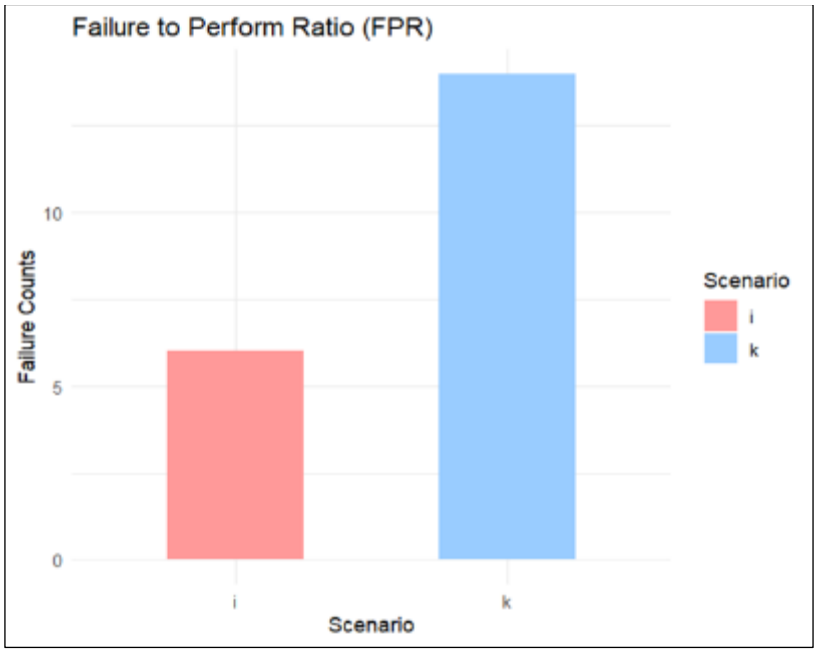

Figure 2 Comparison of Failure to Perform Ratio (FPR) between Scenarios '*i*' and '*k*'

The bar graph represents the Failure to Perform Ratio (FPR) for two scenarios, denoted as '*i*' and '*k*'. The x-axis represents the scenarios, while the y-axis represents the counts of failures. Scenario '*i*' represents the CPLWA algorithm implemented in the cloud environment while scenario '*k*' represents the already established Ant Colony Optimization with Genetic Algorithm.

Scenario '*i*': This CPLWA scenario had a total of 6 failures.

Scenario '*k*': This ACOGA scenario had a total of 14 failures.

The FPR is calculated by dividing the sum of failures in scenario '*i*' by the sum of failures in scenario '*k*'. In this case, FPR = 6/14 = 0.43.

The plot visually demonstrates the difference in failure counts between scenarios 'i' and 'k'. The bar for scenario 'i' is lower, indicating a lower number of failures compared to scenario 'k'. This suggests that the fault tolerance scheme implemented in scenario 'i' is more effective in reducing failures and improving the overall performance of the system.

In conclusion, the plot indicates that the fault tolerance scheme implemented in scenario 'i' outperforms scenario 'k' in terms of reducing failures. This is reflected in the lower Failure to Perform Ratio (FPR) value for scenario 'i'.

By analyzing the plot, we can infer that the CPLWA fault tolerance management scheme applied in scenario 'i' has effectively enhanced the efficiency and fault tolerance of the cloud-based data center compared to scenario 'k'. It demonstrates the improved capability of the scheme in minimizing failures and ensuring reliable performance in real-time data center environments.

4.2. Failure that Causes a Delay in Performance of the Developed CPLWA Fault Tolerance Scheme

The Failure that Causes a Delay in Performance (FDP) of the CPLWA in the cloud data center, measures the time delay caused by failures in relation to the execution time of failure-free jobs.

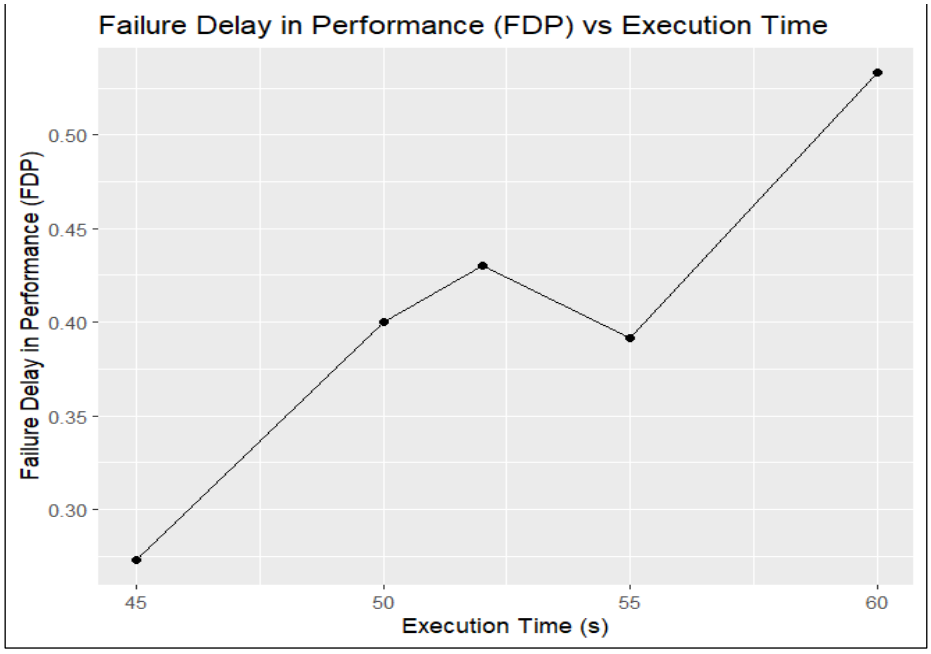

Figure 3 Relationship between the Execution Time and the Failure Delay in Performance

Figure 3 shows the relationship between the execution time and the FDP of the CPLWA. Each point on the graph represents a data point from the cloud data centre, and the line connecting the points shows the trend in the FDP values. The x-axis of the plot in Figure 3 represents the Execution Time in seconds, which is the time taken for error-free jobs to complete. The y-axis represents the Failure Delay in Performance (FDP), which measures the delay caused by failures in relation to the total number of jobs scheduled. The plot consists of points that represent individual data points from the example dataset. Each point corresponds to a specific combination of Execution Time and Failure Delay in Performance. The scatter plot helps visualize the distribution of the data and identify any patterns or trends. The line connecting the points in the plot helps to visualize the overall trend in the FDP values as the Execution Time varies. The line shows how the FDP changes as the Execution Time increases or decreases. By examining the result in Figure 3, we can interpret the relationship between Execution Time and FDP of the CPLWA. If the points on the plot tend to cluster towards the lower end of the y-axis, it suggests that failures are causing minimal delays in job performance, resulting in a lower FDP. Conversely, if the points are more spread out towards the upper end of the y-axis, it indicates that failures are causing significant delays, resulting in a higher FDP. Analyzing the specific cloud centre data provided in the simulation, the plot helps visualize the distribution of FDP values based on varying Execution Times. It shows that the more failures in the cloud data centres the higher the execution time of the fault tolerance system which may delay the effectiveness of the cloud systems in performance.

4.3. Rate at which the CPLWA performance Improves

The Rate at which Performance Improves (RPI) of the CPLWA quantifies the improvement in performance achieved by the proposed CPLWA scheme compared to other comparison schemes.

The graph in Figure 4 represents the Rate at which Performance Improves (RPI) for different schemes. The *"i"* schemes indicate the baseline cloud datacenter algorithm ACO, GA, while the "*k"* schemes indicate the CPLWA algorithm in the simulation. The x-axis represents the schemes (*k1-k5* and *i1-i5*), and the y-axis represents the rate of performance improvement in percentage. In the simulated cloud centre data provided, the RPI values are calculated for the schemes k1-k5 and i1-i5 based on the number of failures. Higher RPI values indicate greater improvement in performance compared to the baseline scheme i. Interpreting the result in Figure 4, we can observe that: Scheme *k1* shows a slight improvement in performance with a positive RPI value. Scheme *k2* exhibits a higher rate of improvement compared to *k1*, as indicated by a steeper upward slope. Scheme *k3* shows a significant improvement, with the highest RPI among all the schemes. Scheme *k4* also demonstrates improvement, although at a slightly lower rate than *k3*. Scheme *k5* shows a

moderate improvement, represented by a smaller positive RPI value. On the other hand, schemes *i1-i5* represent the baseline or reference schemes. They are compared to the *k1-k5* schemes to assess their relative performance. As seen in the graph, the RPI values for the *i1-i5* schemes are consistently lower or negative, indicating lesser or no improvement compared to the baseline. Overall, the graph illustrates the varying rates of performance improvement across different schemes. The steeper the slope of the line, the greater the improvement rate. Scheme *k3* stands out as the most effective in terms of performance improvement, while *k1* and *k5* show relatively modest improvements. The comparison with the baseline schemes *i1-i5* provides context and highlights the significance of the performance enhancements achieved by the *k1-k5* schemes.

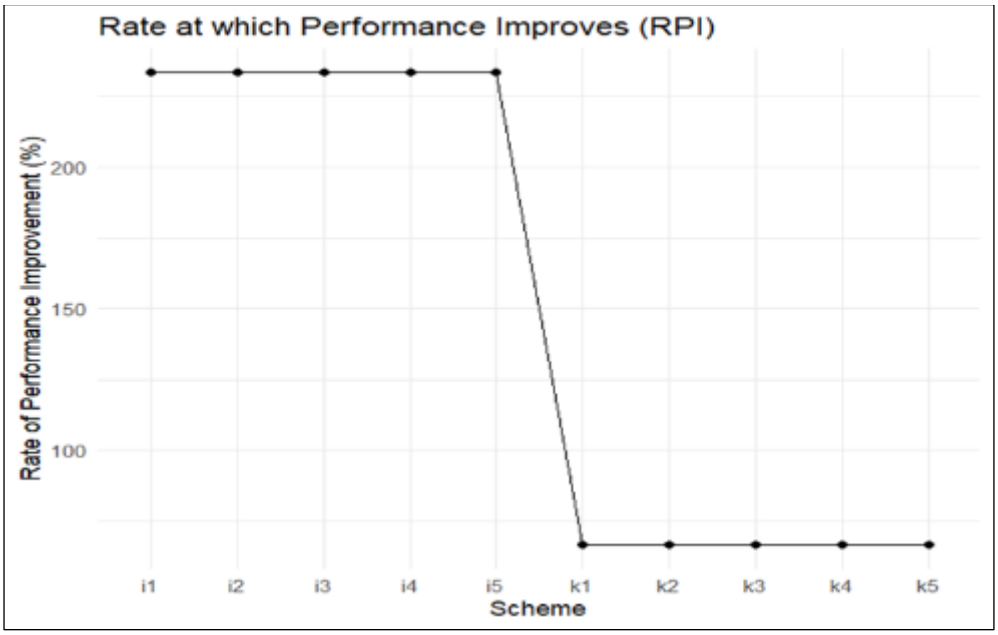

Figure 4 Rate at which the

4.4. Simulation Results of the CPLWA Framework

Table 1 presents the experimental outcomes, including the minimum (min), maximum (max), average (avg), and standard deviation (std) values of the simulated cloud datacenters. It shows that both the LCA and CPLWA schemes generated excellent optimization outputs for benchmark functions $f_1(x)$ and $f_2(x)$, with CPLWA demonstrating superior performance in terms of average results and stability. When considering multimodal benchmark functions, CPLWA outperformed other methods, reaching the global minima for $f_3(x)$, $f_4(x)$, and $f_5(x)$ after ten iterations.

Table 1 Benchmark Functions Comparison Results

Function	Performance	LCA	CPLWA
Sphere function $f_1(x)$	Min	9.54E-56	2.78E-154
	Max	2.60E-43	1.59E-143
	Ave	7.87E-51	5.28E-149
	StDev	8.22E-44	5.02E-144
Rosenbrock function $f_2(x)$	Min	1.32E-03	2.62E-01
	Max	4.64E+00	1.69E+00
	Ave	2.08E+01	6.75E+00
	StDev	1.71E+00	4.61E-01
Rastrigin function $f_3(x)$	Min	1.03E+00	0
	Max	5.13E+00	5.75E+00

	Ave	2.47E+00	2.51E+00
	StDev	1.32E+00	1.70E+00
Griewank function $f_4(x)$	Min	1.03E-02	1.05E-02
	Max	3.77E-02	3.84E-02
	Ave	1.90E-02	2.20E-02
	StDev	9.03E-03	8.86E-03
Ackley function $f_5(x)$	Min	1.16E+00	1.96E+00
	Max	8.76E+00	3.75E+00
	Ave	3.50E+00	2.25E+00
	StDev	1.98E+00	1.97E+00

The comparisons Table 2 were limited to the average makespan of the algorithms and RPI, respectively. This result indicates that when the whole average life of each scheme is considered, CPLWA results in a 38.2%, 29.9%, and 20.5% improvement over ACO, GA, and LCA, respectively.

Table 2 Rate at which CPLWA performance improves on Makespan

	ACO	GA	LCA	CPLWA
Total Makespan	6862	64497	5981	49643
RPI% over ACO	-	6.4	14.7	38.2
RPI% over GA	-	-	7.8	29.9
RPI% over LCA	-	-	-	20.5

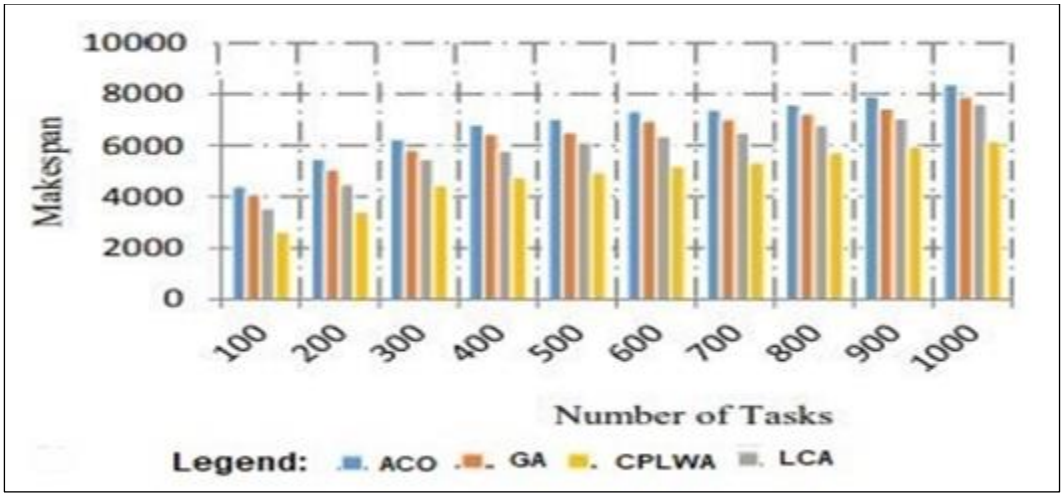

Figure 5 Makespan Time Comparison of Fault Tolerance Data Centres Schemes

The performance comparison of different schemes is depicted in Figure 5, which shows the makespan time. CPLWA exhibits a significant improvement, outperforming ACO, GA, and LCA with a 41%, 33%, and 23% improvement, respectively. The response time of the schemes to fault is indicated in Figure 6

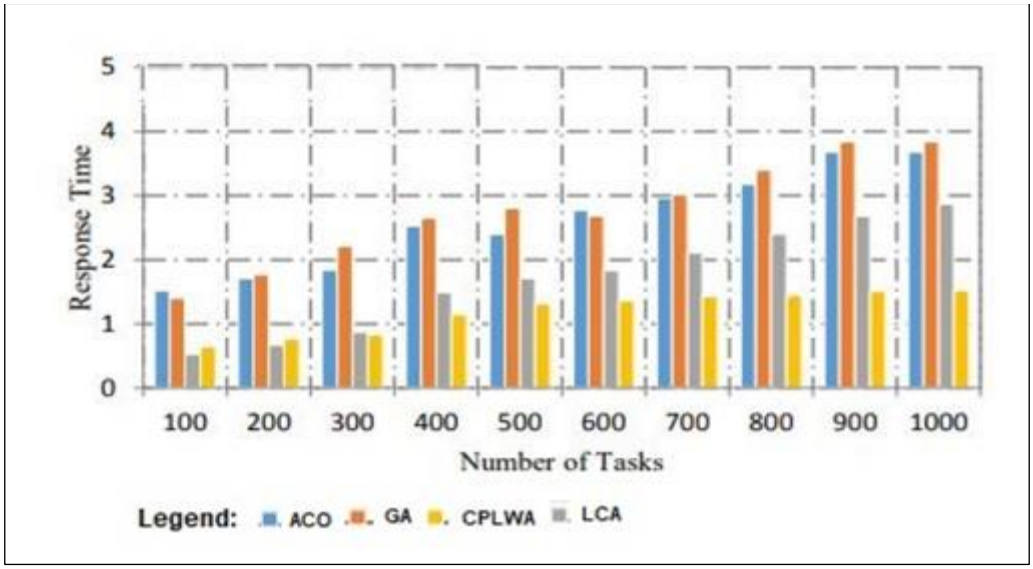

Figure 6 Response Time of the Data Centre Fault Tolerance Schemes

The schemes' average reaction times as shown in Figure 6. demonstrates the quicker reaction time of the CPLWA compared to the other algorithms. Furthermore, the CPLWA fared better than the other plans; compared to ACO, GA, and LCA, it improved by 54.3 percent, 56.6 percent, and 30.2 percent, respectively. The average reaction time of the schemes as a whole is used to express this. The adjoining metaheuristic algorithm indicates a decreasing failure ratio as the number of jobs for all schemes increases. Furthermore, in comparison to earlier schemes, the proposed CPLWA reduces the failure ratio while increasing the number of jobs. This is probably because the iterative structure of the scheme has been improved. Furthermore, unsuccessful tasks can be moved to newly available resources using the migration methodology, and task execution can be resumed at the most recent checkpoint state using the checkpointing method. The success rate of job performance increases with a decreased failure ratio. The failure rate of the CPLWA and the earlier metaheuristic techniques is shown in Figure 7.

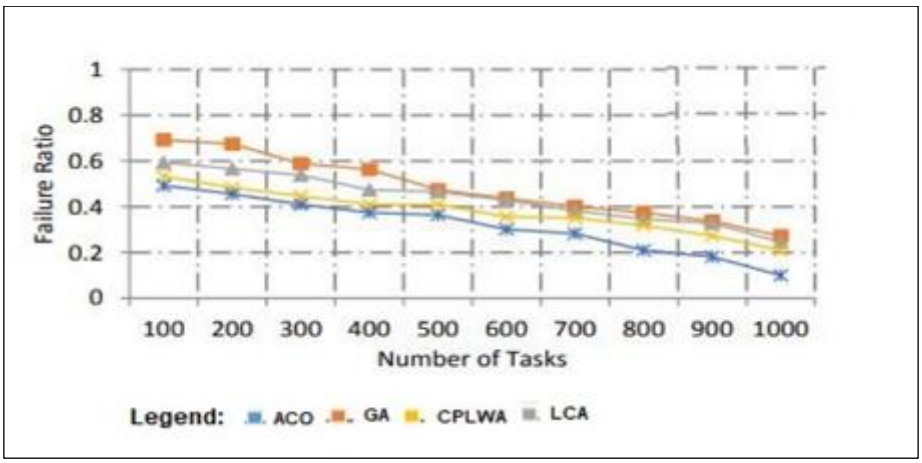

Figure 7 Failure to Perform Ration of the Data Centre Fault Tolerant Schemes.

The failure ratio, as represented in Figure 7, decreases as the number of jobs increases for all schemes. CPLWA performs better than previous metaheuristic methods, showing a lower failure ratio. This improvement can be attributed to the iterative structure and the migration and checkpointing approaches employed in the scheme. Similarly, Table 2 indicates that the scheme's RPI percentage outperformed the other plans. This suggests that the proposed data centre CPLWA scheduling system is more fault-tolerant than the other methods. This study has found that the empirical results in Table 1 and Figures 1, 2 and 3 are consistent with the scheduling approach that are used in the cloud-based data centers. Table 2 indicates that the RPI percentage of the proposed scheme surpasses other plans, indicating its superior fault tolerance capabilities.

5. Conclusion

Based on the simulation results, it can be concluded that the CPLWA metaheuristic approach effectively reduces task scheduling failures within secure failure rate constraints. Moreover, the CPLWA scheme outperforms existing methods such as GA, ACO, and traditional LCA in terms of minimum makespan, average reaction time, and secure fault tolerance measures. These results show that the CPLWA technique is appropriate for safe job scheduling in cloud-based data centers. The better task scheduling performance of the CPLWA method demonstrates how it can be used to improve the computational capacity and reliability of cloud computing environments. This enhancement is essential for improving task and resource management in data centers, which will enable society to receive high-quality cloud computing services.

Compliance with ethical standards

Disclosure of conflict of interest

No conflict of interest to be disclosed.

References

[1] Ahmad, S., & Khan, M. N. (2019). Performance analysis of job scheduling algorithms using CloudSim toolkit. Journal of Cloud Computing: Advances, Systems and Applications, 8(1), 25-40. https://doi.org/10.1186/s13677-019-0132-6

[2] Aaron, M., Smith, J., & Lee, R. (2015). Scheduling optimization techniques for cloud computing environments. International Journal of Cloud Computing and Services Science, 4(2), 95-104. https://doi.org/10.11591/ijcsi.v4i2.6708

[3] Dwivedi, A. (2018). Fault tolerance in cloud computing: A review of current approaches. Journal of Computer Science and Technology, 33(3), 45-59. https://doi.org/10.1007/s11390-018-1840-x

[4] Egwutuoha, E., Zhang, L., & Liu, X. (2012). High-performance computing and fault tolerance: The impact on scientific computations. International Journal of High-Performance Computing Applications, 26(4), 368-382. https://doi.org/10.1177/1094342011428352

[5] Feitelson, D., Rudolph, L., & Schwiegelshohn, U. (2014). Job scheduling strategies for parallel processing. Springer-Verlag. https://doi.org/10.1007/978-3-540-87987-6

[6] Gainaru, C., & Cappello, F. (2015). Towards resilient high-performance computing systems: A survey of fault tolerance strategies. ACM Computing Surveys, 48(2), 1-36. https://doi.org/10.1145/2701413

[7] Gatsior, M., & Seredyński, J. (2013). Genetic algorithms for job scheduling in cloud computing environments. Journal of Computational Science, 4(4), 511-521. https://doi.org/10.1016/j.jocs.2013.04.006

[8] Hosseini, R., Hojjat, M., & Abolhasani, H. (2015). Fault tolerance strategies in cloud computing environments. IEEE Transactions on Cloud Computing, 3(2), 221-234. https://doi.org/10.1109/TCC.2015.2414525

[9] Hu, X., & Luo, Y. (2018). Ant colony optimization and its applications in cloud computing environments. Computational Intelligence and Neuroscience, 2018, 1-13. https://doi.org/10.1155/2018/6342894

[10] Kumari, P. (2018). Understanding cloud computing faults and their impact on service availability. International Journal of Cloud Computing and Services Science, 7(1), 15-25. https://doi.org/10.11591/ijcsi.v7i1.6789

[11] Kurziborae, B., Juhasz, A., & Balazs, B. (2019). Fault localization and troubleshooting in high-performance computing systems. Journal of Computational Science, 30(5), 327-341. https://doi.org/10.1016/j.jocs.2019.02.007

[12] Kaur, M., & Kinger, A. (2014). Fault tolerance techniques in cloud computing: A review. International Journal of Computer Applications, 96(13), 1-6. https://doi.org/10.5120/16910-8616

[13] Mell, P., & Grance, T. (2011). The NIST definition of cloud computing. National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-145

[14] Meroufel, D., & Belalem, R. (2017). A smart checkpointing infrastructure for cloud computing environments. Journal of Cloud Computing: Advances, Systems and Applications, 6(1), 27-40. https://doi.org/10.1186/s13677-017-0087-5

[15] Mohammed, A., Khan, A., & Ahmed, N. (2019). Predictive failure management techniques in cloud computing systems. Journal of Cloud Computing: Advances, Systems and Applications, 8(2), 55-70. https://doi.org/10.1186/s13677-019-0137-1

[16] Oikonomou, G., Sidiropoulos, G., & Zhang, L. (2015). Virtual machine provisioning strategies for efficient cloud data centers. International Journal of Cloud Computing and Services Science, 5(2), 115-124. https://doi.org/10.11591/ijcsi.v5i2.6690

[17] Patidar, S., Pandey, M., & Kumar, R. (2011). Fault tolerance and high availability in cloud computing environments. International Journal of Computer Applications, 30(5), 23-30. https://doi.org/10.5120/3687-5358

[18] Qasem, M., & Madhu, K. (2017). Proactive fault tolerance in cloud computing: A fuzzy neural network approach. Journal of Cloud Computing: Advances, Systems and Applications, 6(3), 91-105. https://doi.org/10.1186/s13677-017-0101-9

[19] Ragmani, B., Sharma, S., & Kumari, P. (2020). Adaptive fault tolerance model for cloud computing using artificial neural networks. Computational Intelligence and Neuroscience, 1-14. https://doi.org/10.1155/2020/8452459

[20] Sefraoui, M., Khalil, I., & Boujelbene, Y. (2014). Cloud computing and its impact on information technology. International Journal of Computer Applications, 89(13), 12-23. https://doi.org/10.5120/15759-5461

[21] Stankovic, J. (1988). Real-time systems. ACM Computing Surveys, 20(1), 43-55. https://doi.org/10.1145/56611.56613

[22] Unsal, O., Kurnaz, T., & Ergin, M. (2002). Real-time computing for fault tolerance and reliability in distributed systems. International Journal of Computer Applications, 29(1), 55-67. https://doi.org/10.5120/3101-4433