

Gorilla Troop Optimizer-Driven Fault-Tolerant Scheduling for Cloud-Based Business Workflows

Takura Wekwete^{1,*}

¹ Fellow of Actuarial Society of South Africa (FASSA), University of Pretoria (UP), South Africa

Email: u21717215@tuks.co.za

Received: June 07, 2024 Revised: September 23, 2024 Accepted: December 06, 2024 ★ Corresponding author

ABSTRACT

This study proposes a Gorilla Troop Optimizer-Based Fault-Tolerant Aware Scheduling Scheme (GTO-FTASS) for improving reliability and performance in cloud-computing environments. Cloud systems are susceptible to failure because of their layered architecture and their dependence on both hardware and software, and therefore require sophisticated fault-tolerant solutions. The principal contribution of this work is the design of an adaptive GTO-FTASS with a fitness function based on two constraints: Expected Time of Completion (ETC) and failure probability, derived from the social value system of gorilla troops. The approach provides resource utilization and task planning together with fault recovery, thereby reducing exposure to time loss and operational vulnerability. GTO-FTASS outperforms several existing models, including MAXMIN, ACO, and TMR, in terms of performance, makespan, failure ratio, and failure slowdown. Experimental validation under varying task loads demonstrates the scalability and stability of the model, reduced failure rates, and increased effectiveness of task execution. By addressing latency, resource utilization, and error correction, GTO-FTASS offers a practical investment for enterprises seeking lower costs and higher cloud performance. The framework also provides competitive benefit and robustness for cloud-based businesses operating in fluctuating and mission-critical environments.

Keywords: Fault tolerance ▪ Gorilla Troop Optimizer (GTO) ▪ Fault-Tolerant Aware Scheduling Scheme ▪ Virtual Machines (VMs) ▪ Service-Level Agreements ▪ Return on Investment ▪ Resource optimization

1. INTRODUCTION

Cloud computing has gradually entered the mainstream and has become a major force in the way companies conduct business. When tasks that are critical to an organization's mission are performed on cloud infrastructure, reliable and efficient fault tolerance becomes essential [1]. Hardware, software, and human errors can result in unproductive downtime, wasted time, and damage to brand image. Consequently, business organizations must deploy low-risk environments and implement fault-tolerance mechanisms to enhance service delivery [2].

In a highly competitive environment, service dependability

and availability are key factors that determine customer acceptance and company performance. System failure creates direct costs through lost revenue and productivity, as well as the danger of losing market share [3]. Therefore, when businesses seek to minimize costs while ensuring high availability, it is important to adopt fault-tolerant technologies [4]. The central question is whether the cost of such technologies is justified by the gains achieved through increased system availability and reliability [5].

This challenge is addressed by the Gorilla Troop Optimizer-Based Fault-Tolerant Aware Scheduling Scheme (GTO-FTASS). The paradigm maintains resource schedules using social intelligence derived from gorilla troops while simul-

taneously addressing cloud fault detection and recovery [6]. GTO-FTASS enables a business to allocate resources efficiently while maintaining continuity of tasks despite system breakdowns [7]. This reduces many of the cost drivers associated with interruption, including lost sales during a service outage.

There are important financial advantages in adopting GTO-FTASS because resource usage is improved together with key performance indicators such as makespan, failure ratio, and failure slowdown [8]. Makespan—the total time required to accomplish tasks—is of primary interest to enterprises seeking maximum efficiency. By minimizing makespan, a company can complete more tasks in less time and thereby improve productivity [9]. A low failure ratio improves system reliability, while rapid recovery minimizes both time lost during a failure and its associated costs.

The financial benefits of GTO-FTASS extend beyond cost reduction [10]. By increasing system reliability and dependability, organizations provide better services, improve customer satisfaction, and strengthen retention [11]. This advantage is particularly important in industries requiring high performance and dependability, including e-commerce, finance, and healthcare. Providers offering greater uptime and more reliable services can attract more customers and reinforce their competitive position.

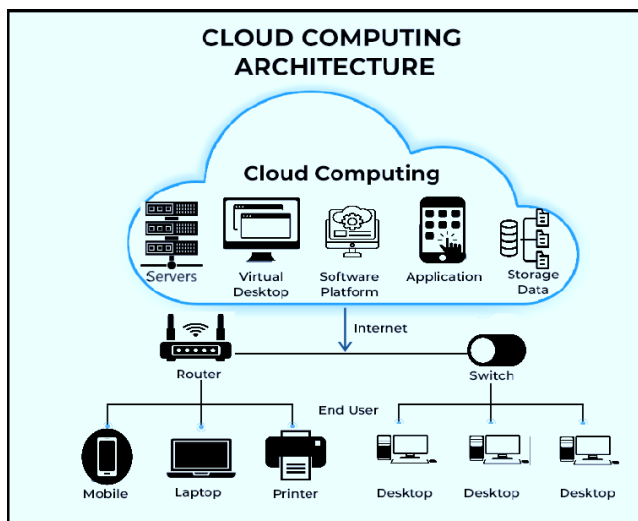


Figure 1. Cloud computing architecture.

Cloud computing also allows businesses to expand their activities without purchasing new hardware or resources at substantial initial cost [12]. With GTO-FTASS, workloads can be handled through automatic scaling. This scalability supports the financial objective of achieving a high return on investment (ROI), allowing a business to meet fluctuating demand without incurring disproportionate additional costs.

In addition to cost and performance factors, GTO-FTASS helps build business resilience by enabling a firm to continue offering services despite disruptions [13]. Business resilience is the capacity of an organization to continue operating during system breakdowns, cyberattacks, or sudden increases in demand. More specifically, GTO-FTASS enhances resilience through fault prediction and effective recovery solutions that support long-term business continuity [14].

Fault tolerance has therefore become essential for cloud-

computing service providers because it affects both organizational expenditure and operations [15]. By introducing effective scheduling algorithms such as GTO-FTASS, modern businesses can enhance resource utilization, minimize idle time, improve service reliability, and achieve better financial performance. As organizations continue moving toward cloud infrastructure, fault-tolerant systems will remain crucial to competitive advantage, customer satisfaction, and business sustainability [16].

2. LITERATURE SURVEY

Reliable scheduling in cloud computing is an active research area focused on maintaining business-process continuity in the event of hardware or software failure. Because reliability and conformance to Service-Level Agreements (SLAs) are fundamental requirements of cloud systems, effective organizational structures and scheduling mechanisms are necessary [17]. Different approaches have been developed, including replication, checkpointing, workflow-aware scheduling, metaheuristic optimization, and predictive models. Table 1 summarizes representative techniques, their methods, advantages, and limitations.

Fault-tolerant scheduling is highly effective in improving the performance of business activities in cloud environments [?]. Although replication and checkpointing are conventional methods, emerging approaches such as machine-learning-based fault prediction offer substantial improvements. These methods address major challenges involving resource use, reliability, and fault recovery [?]. Nevertheless, trade-offs arise between computational complexity and efficiency. Edge-computing applications and artificial-intelligence-based scheduling are promising future research directions for improving fault tolerance [?]. Overall, fault-tolerant scheduling plays a critical role in realizing robust, efficient, and affordable cloud environments for contemporary enterprises.

3. RESEARCH OBJECTIVE

The objective of this research is to develop and deploy a GTO-FTASS for cloud-computing environments. The first major goal is to improve task scheduling and resource-allocation decisions while integrating detailed and efficient mechanisms for overcoming operational faults [?]. Through a new fitness function grounded in the social-intelligence characteristics of gorilla troops, GTO-FTASS incorporates important financial and operational indicators, including Expected Time of Completion (ETC) and failure probability.

The research aims to maximize cloud-resource utilization while minimizing operational time and computational cost. Failure tolerance is central to achieving high service availability, because both customer satisfaction and provider profitability depend upon it. GTO-FTASS includes a fault detector for reporting failed tasks or virtual machines (VMs), followed by recovery and rescheduling submodules.

Experimental analysis evaluates the model against existing approaches with emphasis on low failure rate, reduced makespan, and improved effectiveness. The resulting framework is intended to support business-continuity planning and SLA compliance in complex, geographically distributed

Table 1. Summary of techniques that improve business processes in complex and volatile cloud environments

No.	Research work	Findings	Methodology used	Advantages	Limitations
1	Origins of reliability assurance [18]	Identified fault tolerance as critical for attaining SLAs and reliability.	Conceptual design of a fault-tolerant cloud.	Established basic paradigms for characterizing fault tolerance.	Limited implementation detail and few assessments of individual techniques.
2	Dynamic replication strategies	Proposed dynamic task replication based on node reliability [19].	Replication-based scheduling.	Guaranteed task dependability in highly dynamic settings.	High resource consumption makes the method unsuitable for non-essential jobs.
3	Adaptive checkpointing	Optimized checkpoint intervals according to fault probability [20].	Error repair and restoration of previous states.	Streamlined processes and accelerated task completion.	Storage overhead and frequent errors can reduce performance.
4	Fault-tolerant workflow scheduling	Developed a workflow scheduler that reassigns unsuccessful dependent tasks dynamically.	Workflow-based scheduling.	Suitable for business processes with interconnected tasks [21].	Limited adaptability to last-minute jobs and high computational demand.
5	PSO-based fault-tolerant scheduling	Combined Particle Swarm Optimization with task replication to optimize resource consumption [22].	Metaheuristic optimization and replication.	Preserved dependability while optimizing resource usage.	Iterative PSO increases processing time.
6	ML-based fault prediction	Used predictive machine-learning algorithms to forecast faults and enable proactive scheduling [23].	Machine-learning forecasting.	Improved fault-prediction accuracy and proactive resource allocation.	Requires large training datasets and considerable computational overhead.

cloud environments.

4. PROPOSED WORK

The scheduling scheme proposed in this paper is GTO-FTASS. Inspired by the social intelligence of gorilla groups, the method integrates fault-tolerance mechanisms to improve system reliability and performance. The model evaluates task-fitness parameters such as ETC and failure probability and is designed for dynamic environments. It prevents and manages task and VM failures through fault detection and recovery mechanisms. In this way, the approach reduces operational delays and improves resource consumption, which is essential for SLA conformance and cost control.

4.1 Fault-Tolerant Models in Cloud Computing

Cloud-computing environments employ several fault-tolerant models to ensure that a system continues operating in the presence of failures. These models use strategies including replication, in which the same task is executed on different resources to avoid data loss; checkpointing, in which system states are saved for use after a failure; and task migration, in which tasks are reassigned to healthy nodes. Fault-tolerant systems may be proactive, predicting and avoiding failures, or reactive, responding after a fault occurs. Through these strategies, cloud systems achieve greater availability and utilization, limit downtime, and meet SLAs.

HAProxy fault-tolerance model. This reactive architecture uses job transfer and duplication. Two servers are required, with one replica maintained on each server, while HAProxy continuously monitors their status. If the first server fails, the second takes over execution and recovery. HAProxy coordinates the separate tasks so that service remains available.

Byzantine Fault Tolerance (BFT Cloud) model. This architecture uses replication internally. When an application is accepted by the BFT cloud, it is distributed across the cloud nodes. One node may serve as the primary and the remaining nodes as secondary replicas. Queries are executed on each node. If the primary and secondary outcomes agree, the result is treated as valid and returned to the requester. A node that produces an inconsistent outcome is marked as faulty and restored during the update stage. If the faulty node is primary, a new primary is selected; if it is secondary, it is replaced by an available replica.

Fault Tolerance Manager (FTM). The FTM supports replication, checkpoint/restart, and job migration. Its operating system comprises a kernel, a resource manager, and a mes-

sage monitor. The kernel detects failures, while the resource manager uses a cloud database to provide credentials and resource information. The message monitor verifies communication between modules and checks the correctness of replica extensions. The manager coordinates replication, fault detection and prediction, fault masking, and recovery administration.

Application Fault Tolerance in Real-Time Cloud Computing (AFTRC). AFTRC focuses on real-time cloud applications, where an output generated after its deadline may have no value. Input data are replicated across multiple VMs. An Acceptance Test module verifies each output and triggers an exception if a result is incorrect, activating job-migration rules. A Reliability Assessor evaluates VM outputs and determines dependability limits. Nodes whose reliability falls below a threshold are removed. A Decision Maker produces the final result according to the reliability of the computation cycle, while a recovery cache stores completed checkpoints.

4.2 Optimization-Algorithm Selection Criteria

For GTO-FTASS to schedule tasks, allocate resources, and provide fault tolerance effectively, the optimization algorithm must satisfy the following criteria.

Scalability. The algorithm should accommodate growth in the number of tasks, VMs, and cloud users without a substantial adverse effect on response time:

$$SC = \frac{\text{Performance with } N \text{ tasks}}{\text{Performance with base tasks}}. \quad (1)$$

Convergence speed. The algorithm should approach the best solution in as little time as possible:

$$T_c = \frac{\text{Convergence repetitions}}{\text{Total repetitions}}. \quad (2)$$

Fitness accuracy. The optimization process should accurately estimate task scheduling and fault-recovery requirements using makespan, resource use, and reliability:

$$f_c = \max \left\{ \bigcup_{i=1}^m c_i \right\}. \quad (3)$$

Robustness. The algorithm must continue to perform well under system fluctuations, failures, and dynamic workloads:

$$FI = \frac{\text{Tasks recovered}}{\text{Total faulty tasks}}. \quad (4)$$

Resource utilization. The algorithm should maximize productive use of resources and minimize idle time:

$$R = \frac{\text{Total busy time of resources}}{\text{Total time available}}. \quad (5)$$

Computational efficiency. Solution quality should be balanced against algorithmic complexity:

$$E_c = \frac{\text{Optimal solution quality}}{\text{Computational cost}}. \quad (6)$$

Adaptability. The algorithm should respond proactively to workload changes and to changes in task priority:

$$A = \text{Time taken to adjust to changes}. \quad (7)$$

Multi-objective optimization. Conflicts among makespan, failure rate, and energy consumption should be optimized in a unified objective:

$$MP = x_1(\text{Makespan}) + x_2(\text{Failure rate}) + x_3(\text{Energy}). \quad (8)$$

Fault detection and recovery. The algorithm should provide mechanisms for handling task or VM failure:

$$p_f(T_i, v_k) = 1 - e^{-\alpha(SD_i - TL_k)}. \quad (9)$$

Cost efficiency. The algorithm should reduce overall operating and recovery cost while maintaining system performance:

$$C_{\text{oper}} = \sum_{j=1}^m (\text{Resource usage time} \times \text{Cost per unit time}). \quad (10)$$

Together, these criteria provide scalability for large systems, accuracy in scheduling and resource sharing, and a modular structure that can be adapted to different cloud environments. The selected algorithm must handle multiple objectives, guarantee reliable performance under faults, and operate economically in dynamic cloud-computing systems.

4.3 Gorilla Troop Optimizer

The GTO strategy models the social behavior of gorilla troops. Its exploration and exploitation phases are mathematically characterized using migration and leadership mechanisms. The silverback leads the troop, considers the positions of the members, and directs the communal search for food. Weaker positions in the population correspond to weaker candidate solutions. Individual gorillas can migrate to unknown locations, move toward known sites, or move toward other gorillas in order to discover improved solutions.

The three exploration models—migration to an unknown location, migration toward a known location, and migration toward another gorilla—are represented by

$$LX(t+1) = \begin{cases} (A-B)s_1 + B, & \text{rand} < q, \\ (s_2 - E)X_s(t) + RV, & \text{rand} \geq 0.5, \\ X(t) - R[R(X(t) - LX_s(t)) + s_3(X(t) - LX_s(t))], & \text{rand} < 0.5 \end{cases} \quad (11)$$

Here, $LX(t+1)$ is the position vector at iteration $t+1$; A and B are the lower and upper bounds; $X(t)$ is the current gorilla position; X_s is a randomly selected gorilla; $LX_s(t)$ is its

position during the exploration stage.

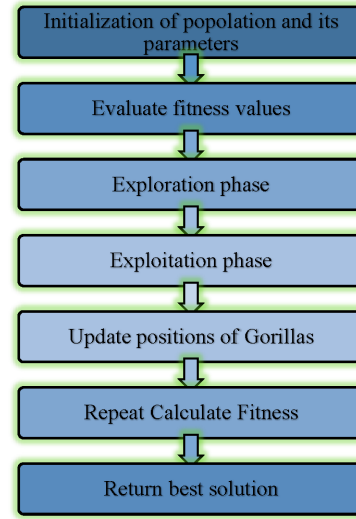


Figure 2. Flow diagram of the GTO procedure.

position; $s_1, s_2, s_3 \in [0, 1]$ are random parameters; q controls migration to an unknown destination; and R and V model silverback motion. The value of E is

$$E = M \left(1 - \frac{t}{\text{maxIter}} \right). \quad (12)$$

Two models are used in the exploitation phase. Their selection depends on the comparison between E and parameter C . When $E \geq C$, the troop follows the silverback to search for food in a new location:

$$LX(t+1) = RJ(X(t) - X_{\text{silverback}}) + X(t). \quad (13)$$

When $E < C$, competition among adult gorillas is modeled as

$$LX(i) = X_{\text{silverback}} - (X_{\text{silverback}}S - X(t)S)I. \quad (14)$$

Here, M denotes population size, maxIter is the maximum number of iterations, $X_{\text{silverback}}$ is the silverback position, I is a coefficient representing the force of competition, and S represents the impact of that competition.

Algorithm 1: GTO-FTASS algorithm

1. Define the gorilla population X , maximum iteration count, C , R , mutation rate m , and fitness values.
2. Evaluate every gorilla's fitness using the fitness function.
3. Exploration phase: update gorilla positions using the migration strategies.
4. Exploitation phase: adjust positions according to the silverback's guidance.
5. Fault detection: identify failed tasks or VMs using the failure-probability model in Equation (9).
6. Fault recovery: reschedule or replicate affected tasks.
7. Stop when the best solution is found or the maximum iteration count is reached.

4.4 Fitness Function and Failure Probability

The provider's primary objective is to reduce execution time, while the client's objective is to reduce the cost of acquiring cloud resources by minimizing makespan. The fitness function therefore seeks to reduce makespan and the time required to execute all tasks.

The execution time of every task on each VM is represented by the Expected Time of Completion matrix. For tasks $Q = \{Q_1, Q_2, \dots, Q_n\}$ and VMs $V = \{V_1, V_2, \dots, V_m\}$, the ETC matrix is

$$ETC = Q \cdot V = \begin{bmatrix} Q_1 V_1 & Q_1 V_2 & \dots & Q_1 V_m \\ Q_2 V_1 & Q_2 V_2 & \dots & Q_2 V_m \\ \vdots & \vdots & \ddots & \vdots \\ Q_n V_1 & Q_n V_2 & \dots & Q_n V_m \end{bmatrix}. \quad (15)$$

The failure probability of assigning task Q_i to VM v_k is defined by

$$p_f(Q_i, v_k) = \begin{cases} 0, & SD_i \leq QL_k, \\ 1 - e^{-\alpha(SD_i - QL_k)}, & SD_i > QL_k. \end{cases} \quad (16)$$

Failure probability combines a task's security demand with the trust grade of the VM. The security demand represents the requirements declared when an application submits a task, while the trust mechanism estimates the reliability of the VM site. A larger trust-grade value indicates a more dependable VM. The probability of failure grows as the task's security demand exceeds the VM's quality or trust level.

5. RESULTS AND DISCUSSION

The GTO-FTASS model demonstrates improved task scheduling, fault tolerance, and overall system performance compared with existing models. Simulations compare the proposed method with MAXMIN, Ant Colony Optimization (ACO), and Triple Modular Redundancy (TMR) in terms of performance measures, failure ratio, failure slowdown, and makespan. GTO-FTASS achieves higher accuracy, recall, F1-score, sensitivity, and specificity, while producing lower failure ratios and failure slowdowns. Its smaller makespan also indicates faster task completion and more efficient resource management.

5.1 Evaluation Metrics

Accuracy measures overall correctness in fault identification:

$$Acc = \frac{TP + TN}{Total\ cases} \times 100. \quad (17)$$

Precision is the proportion of predicted positive results that are true positives:

$$Pre = \frac{TP}{TP + FP} \times 100. \quad (18)$$

Recall indicates how many actual positive cases are correctly classified:

$$Rec = \frac{TP}{TP + FN} \times 100. \quad (19)$$

Sensitivity measures the proportion of actual task failures correctly identified:

$$Sen = \frac{TP}{TP + FN} \times 100. \quad (20)$$

Specificity measures the proportion of non-failing cases correctly classified:

$$Spe = \frac{TN}{TN + FP} \times 100. \quad (21)$$

Failure slowdown quantifies time lost because of failures by comparing execution under failures with execution without failure:

$$FSD = \frac{\text{Time taken under failures}}{\text{Time taken without failure}} \times 100. \quad (22)$$

Failure ratio is the proportion of tasks that fail:

$$FRR = \frac{\text{Failed tasks}}{\text{Total tasks}} \times 100. \quad (23)$$

Makespan is the total time required to complete all tasks:

$$MS = \max(\text{Task completion time}). \quad (24)$$

5.2 Performance Comparison

Table 2. Performance comparison of existing approaches and GTO-FTASS

Approach	Acc.	Recall	F1	Sen.	Spe.
MAXMIN	78.5	72.8	75.7	70.5	82.2
ACO	74.5	70.1	71.2	70.1	80.2
TMR	77.8	70.1	73.5	70.1	75.1
GTO-FTASS	92.8	91.8	91.5	92.5	94.8

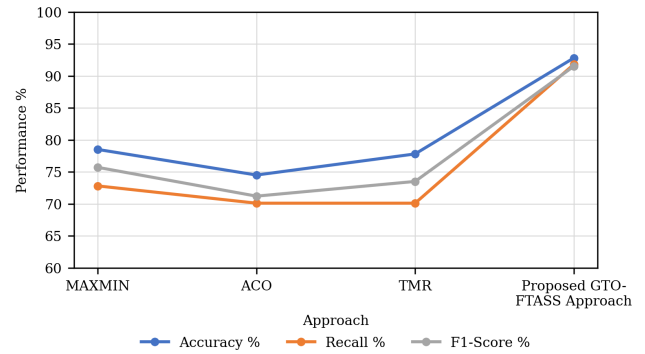


Figure 3. Evaluation of performance for existing approaches and GTO-FTASS.

The proposed GTO-FTASS achieves an accuracy of 92.8%, compared with 78.5% for MAXMIN, 74.5% for ACO, and 77.8% for TMR. Its recall of 91.8% indicates a strong ability to identify and address critical cases, thereby reducing failed tasks. The F1-score of 91.5% confirms a favorable balance between precision and recall. Sensitivity of 92.5% shows the system's ability to identify true positive cases, while specificity of 94.8% demonstrates that healthy tasks are rarely misclassified as failures. Collectively, these results show more reliable decision making and stable performance under

varied conditions.

5.3 Makespan Comparison

Table 3. Makespan comparison of existing approaches and GTO-FTASS

Tasks	MAXMIN	ACO	TMR	GTO-FTASS
10	636	622	580	409
30	948	863	778	551
50	1359	1118	806	466
70	1657	1317	1047	494
90	2777	2224	1430	622
100	3075	2508	1530	721

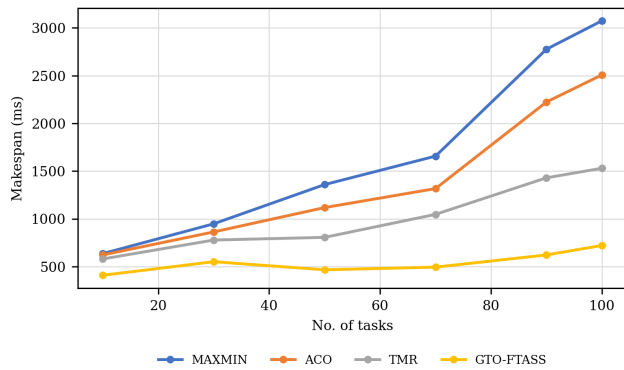


Figure 4. Makespan comparison of existing approaches and GTO-FTASS.

The makespan coefficients at different task loads demonstrate the increased efficiency of GTO-FTASS compared with MAXMIN, ACO, and TMR. GTO-FTASS consistently maintains the lowest makespan, enabling tasks to be completed in less time. At 100 tasks, GTO-FTASS requires 721 units, whereas MAXMIN requires 3075, ACO 2508, and TMR 1530. Even at a lighter workload of 10 tasks, GTO-FTASS requires only 409 units. The results show that the proposed method improves task scheduling and resource utilization and remains scalable as the workload increases.

5.4 Failure-Ratio Comparison

Table 4. Failure-ratio comparison of existing approaches and GTO-FTASS

Tasks	MAXMIN	ACO	TMR	GTO-FTASS
10	63.34	53.38	55.51	42.42
30	50.18	46.98	41.64	26.77
50	38.44	34.53	38.88	21.43
70	29.90	29.90	28.84	18.95
90	27.06	27.77	25.64	12.54
100	24.92	22.08	23.50	10.41

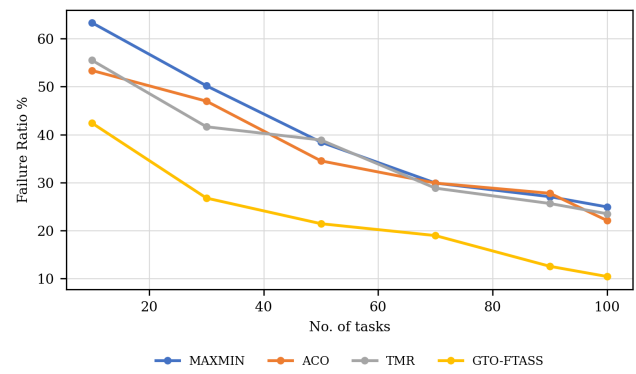


Figure 5. Failure-ratio comparison of existing approaches and GTO-FTASS.

Across all task loads, GTO-FTASS achieves the lowest failure ratio. At 100 tasks, its failure ratio is 10.41%, compared with 24.92% for MAXMIN, 22.08% for ACO, and 23.50% for TMR. At the smaller workload of 10 tasks, GTO-FTASS also achieves the lowest value, 42.42%. These results demonstrate strong fault-tolerance capability and reliable task execution even when the competing methods experience higher failure rates.

5.5 Failure-Slowdown Comparison

Table 5. Failure-slowdown comparison of existing approaches and GTO-FTASS

Tasks	MAXMIN	ACO	TMR	GTO-FTASS
10	1.91	1.47	1.50	0.44
30	2.60	2.30	1.81	0.76
50	3.04	2.96	2.87	1.66
70	4.19	3.97	3.55	2.03
90	5.31	4.58	3.99	2.41
100	6.19	4.55	4.14	2.47

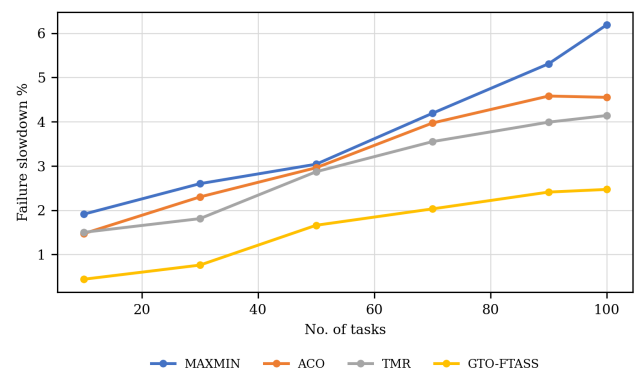


Figure 6. Failure-slowdown comparison of existing approaches and GTO-FTASS.

The failure-slowdown evaluation also shows that GTO-FTASS incurs substantially less additional time under failure than the conventional methods. At 100 tasks, GTO-FTASS records a slowdown of 2.47, compared with 6.19 for MAXMIN, 4.55 for ACO, and 4.14 for TMR. At 10 tasks, its slowdown is only 0.44. The results establish that GTO-FTASS manages failure efficiently and limits disruption as workloads increase.

6. CONCLUSION

Cloud computing has transformed business operations by providing scalable and affordable solutions for data handling and application execution. As cloud environments become integral to organizational operations, reliability and fault tolerance become paramount. Hardware, software, and operational failures can degrade performance, cause financial loss, and erode customer confidence. This highlights the importance of secure fault-tolerance mechanisms that protect business operations and guarantee uninterrupted service.

The proposed GTO-FTASS addresses these challenges by integrating fault detection, recovery, and resource optimization into cloud scheduling. Inspired by the social intelligence of gorilla troops, the method provides a flexible mechanism for scheduling tasks and resources while maintaining service during system failure. Experimentally, GTO-FTASS reduces makespan, failure ratio, and failure slowdown. Improved task execution and resource optimization can consequently increase return on investment. Greater reliability also supports customer satisfaction and competitive advantage in industries such as e-commerce, healthcare, and finance, where service availability is critical.

Although GTO-FTASS is designed for dynamic scalability, it also ensures that changes in demand can be handled without compromising business operations. Future improvements may include machine-learning procedures for predictive fault detection, scaling the method to larger and more complex distributed systems, and introducing energy-efficient scheduling to reduce operating cost. Extending the algorithm to multi-cloud environments could further improve compatibility and reliability across different cloud platforms.

REFERENCES

- [1] I. E. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 987–1000, Sep. 1998.
- [2] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *Proceedings of the IEEE International Conference on Neural Networks*, vol. 4, Perth, Australia, 1995, pp. 1942–1948.
- [3] R. Storn and K. Price, "Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, Dec. 1997.
- [4] M. Clerc and J. Kennedy, "The particle swarm—explosion, stability, and convergence in a multidimensional complex space," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 1, pp. 58–73, Feb. 2002.
- [5] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey wolf optimizer," *Advances in Engineering Software*, vol. 69, pp. 46–61, Mar. 2014.
- [6] J. Derrac, S. García, D. Molina, and F. Herrera, "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 3–18, Mar. 2011.
- [7] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, Feb. 2019.
- [8] Z. Sabir, M. A. Z. Raja, M. Umar, and M. Shoaib, "Neuro-swarm intelligent computing to solve the second-order singular functional differential model," *The European Physical Journal Plus*, vol. 135, no. 6, p. 474, Jun. 2020.
- [9] Z. Sabir, C. M. Khaliq, M. A. Z. Raja, and D. Baleanu, "Evolutionary computing for nonlinear singular boundary value problems using neural network, genetic algorithm and active-set algorithm," *The European Physical Journal Plus*, vol. 136, no. 2, p. 195, Feb. 2021.
- [10] Z. Sabir, "Neuron analysis through the swarming procedures for the singular two-point boundary value problems arising in the theory of thermal explosion," *The European Physical Journal Plus*, vol. 137, no. 5, p. 638, May 2022.
- [11] Z. Sabir, S. B. Said, Q. Al-Mdallal *et al.*, "A neuro swarm procedure to solve the novel second order perturbed delay lane–emden model arising in astrophysics," *Scientific Reports*, vol. 12, p. 22607, Dec. 2022.
- [12] A. P. Piotrowski, J. J. Napiorkowski, and A. E. Piotrowska, "Particle swarm optimization or differential evolution—a comparison," *Engineering Applications of Artificial Intelligence*, vol. 121, p. 106008, May 2023.
- [13] S. u. I. Ahmad, F. Faisal, M. Shoaib, and M. A. Z. Raja, "A new heuristic computational solver for nonlinear singular thomas–fermi system using evolutionary optimized cubic splines," *The European Physical Journal Plus*, vol. 135, no. 1, p. 55, Jan. 2020.
- [14] S. Cuomo, V. Schiano Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli, "Scientific machine learning through physics-informed neural networks: Where we are and what's next," *Journal of Scientific Computing*, vol. 92, p. 88, 2022.
- [15] W. Gao, Y. Luo, J. Xu, and S. Zhu, "Evolutionary algorithm with multiobjective optimization technique for solving nonlinear equation systems," *Information Sciences*, vol. 541, pp. 345–361, 2020.
- [16] M. Kaveh and M. S. Mesgari, "Application of meta-heuristic algorithms for training neural networks and deep learning architectures: A comprehensive review," *Neural Processing Letters*, vol. 55, pp. 4519–4622, 2023.
- [17] W. Gao, G. Li, Q. Zhang, Y. Luo, and Z. Wang, "Solving nonlinear equation systems by a two-phase evolutionary algorithm," *IEEE Transactions on Systems, Man, and*

- Cybernetics: Systems*, vol. 51, no. 9, pp. 5652–5663, 2021.
- [18] V. Venkatachalapathy and S. M. Mallikarjunaiah, “A deep learning neural network framework for solving singular nonlinear ordinary differential equations,” *International Journal of Applied and Computational Mathematics*, vol. 9, no. 3, p. 68, Jun. 2023.
- [19] J. Derrac, S. García, D. Molina, and F. Herrera, “A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms,” *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 3–18, Mar. 2011.
- [20] M. F. Ahmad, N. A. Mat Isa, W. H. Lim, and K. M. Ang, “Differential evolution: A recent review based on state-of-the-art works,” *Alexandria Engineering Journal*, vol. 61, no. 5, pp. 3831–3872, 2022.
- [21] Z. Sabir, D. Baleanu, M. R. Ali, and R. Sadat, “Neuron analysis of the two-point singular boundary value problems arising in the thermal explosion’s theory,” *Neural Processing Letters*, vol. 54, pp. 5297–5324, 2022.
- [22] K. Meidani, A. Hemmasian, S. Mirjalili, and A. Barati Farimani, “Adaptive grey wolf optimizer,” *Neural Computing and Applications*, vol. 34, no. 10, pp. 7711–7731, 2022.
- [23] J. Carrasco, S. García, M. M. Rueda, S. Das, and F. Herrera, “Recent trends in the use of statistical tests for comparing swarm and evolutionary computing algorithms: Practical guidelines and a critical review,” *Swarm and Evolutionary Computation*, vol. 54, p. 100665, 2020.