Intelligent Automation & Soft Computing
DOI: 10.32604/iasc.2023.029495
Article

Tech Science Press

Gorilla Troops Optimizer Based Fault Tolerant Aware Scheduling Scheme for Cloud Environment

R. Rengaraj alias Muralidharan[1,*] and K. Latha[2]

[1]Department of Information Technology, Saranathan College of Engineering, Tiruchirapalli, 620012, Tamilnadu, India
[2]Department of Computer Science and Engineering, Anna University (BIT Campus), Trichy, 620024, Tamilnadu, India
*Corresponding Author: R. Rengaraj. Email: rengaraj-it@saranathan.ac.in
Received: 04 March 2022; Accepted: 12 April 2022

Abstract: In cloud computing (CC), resources are allocated and offered to the clients transparently in an on-demand way. Failures can happen in CC environment and the cloud resources are adaptable to fluctuations in the performance delivery. Task execution failure becomes common in the CC environment. Therefore, fault-tolerant scheduling techniques in CC environment are essential for handling performance differences, resource fluxes, and failures. Recently, several intelligent scheduling approaches have been developed for scheduling tasks in CC with no consideration of fault tolerant characteristics. With this motivation, this study focuses on the design of Gorilla Troops Optimizer Based Fault Tolerant Aware Scheduling Scheme (GTO-FTASS) in CC environment. The proposed GTO-FTASS model aims to schedule the tasks and allocate resources by considering fault tolerance into account. The GTO-FTASS algorithm is based on the social intelligence nature of gorilla troops. Besides, the GTO-FTASS model derives a fitness function involving two parameters such as expected time of completion (ETC) and failure probability of executing a task. In addition, the presented fault detector can trace the failed tasks or VMs and then schedule heal submodule in sequence with a remedial or retrieval scheduling model. The experimental validation of the GTO-FTASS model has been performed and the results are inspected under several aspects. Extensive comparative analysis reported the better outcomes of the GTO-FTASS model over the recent approaches.

Keywords: Cloud computing; gorilla troops optimizer; task scheduling; fault tolerant; task completion time; failure probability

1 Introduction

Over the past decade, Cloud computing (CC) is emerging as a prominent paradigm and its usage had witnessed considerable development [1]. Small scale users and largescale commercial businesses and scientific applications get benefitted from using cloud. By the use of minimum involvement, clients can benefit service from CC since it allows universal and on demand access to a common pool of CC resources which can be hardware, software, and applications are communal resources. Fault might take place on each of these layers; nonetheless, software enabled algorithm is recognized and employed to

recuperate from faults. Fault tolerance is defined as a capability of a system to keep on running its planned operation against faults or errors [2,3]. Even a well-developed system with the highest services and components could not be named reliable with non-fault tolerant abilities [4]. For the reason that a great amount of delay aware applications have to be performed, reliability is a crucial feature of CC. Moreover, service reliability is crucial to cloud wide recognition. Consequently, fault tolerant approaches have gained considerable attention from the researcher. There are different fault tolerance models—checkpointing, Reconfiguration, replication, Safety-Bag Checks, Task Migration, Retry, Task Resubmission, Self-Healing, Masking, and so on. [5–8]—to address the fault at different levels whether reactively or proactively. CC includes the dynamic resource allocation and the usage of data center that is often distributed geographically. The splitting of the server accessible resource into virtual machine (VM) take place by VM monitor (VMM). Multiple or individual VM are allocated for running the provided applications based on client requests. The advantage of using a VM is that it enables us to perform applications on different IDEs, software environments, and operating systems. Fault tolerance based on hardware or time redundancy to cover component or task failure. Time redundancy includes the re-execution of failed task afterward the malfunction was recognized. It is enhanced by the application of checkpoint technique however, still it imposes a significant latency. These two technique presents negative and downside aspects, replication needs additional resources and re-execution need additional time, especially energy. This makes a tradeoff between hardware and time redundancy, in IaaS-CC system replication is preferred mainly since the response time is usually very vital [9,10]. This study focuses on the design of Gorilla Troops Optimizer Based Fault Tolerant Aware Scheduling Scheme (GTO-FTASS) in CC environment in order to schedule the tasks and allocate resources by considering fault tolerance into account. The GTO-FTASS algorithm is based on the social intelligence nature of gorilla troops. Besides, the GTO-FTASS model derives a fitness function involving two parameters such as expected time of completion (ETC) and failure probability of executing a task. In addition, the presented fault detector can trace the failed tasks or VMs and later schedule healing submodules in sequence with a remedial or retrieval scheduling model. The experimental validation of the GTO-FTASS model has been performed and the results are inspected under several aspects.

2 Related Works

This section reviews the recently developed models for task scheduling with fault tolerance in CC. Kasu et al. [11] presented a bloom filter enabled data-aware probability based fault tolerant (DAFT) method that might deal with these faults. Also, presented a data and layout aware method for fault tolerance (DLFT) to efficiently deal with the false positives match. Kanwal et al. [12] presented genetic algorithm (GA) based multi-phase fault tolerance (MFTGA) method for scheduling the task through the VM for multiple users. The presented method effectively maps optimum VM with users based on the service level agreement (SLA). The proposed technique encompasses four stages such as local phase, individual phase, fault tolerance phase, and global phase. Sathiyamoorthi et al. [13] present an adaptive and effective fault tolerant scheduling method to help error-free scheduling. The presented technique considered the highly effective variables namely present workload and failure rate of the resource for optimum Quality of Service (QoS). The presented technique can be validated by utilizing the CloudSim toolkit. Jing et al. [14] proposed a QoS scheduling algorithm by integrating the cloud features, later, we developed a task scheduling object to guarantee fault can be tolerated at the time of the task execution. At last, we presented a QoS-aware scheduling approach, QoS-DPSO, for satisfying the QoS requirement in CC system. Alaei et al. [15] presented an adaptive fault detector approach based Improved Differential Evolution (IDE) method in CC that could reduce the energy utilization, the make-span, overall costs, and, simultaneously, tolerate up fault while scheduling scientific tasks. This presented method employs an ANFIS predictive method for proactively controlling resource load fluctuation which improves the fault

predictive performance beforehand of failure or fault existence. Karthikeyan et al. [16] established a SALDEFT model for improving the communication overhead (that is, overall energy utilization and the network resource). Now, choosing an optimum Physical Machine (PM) is taken into account as an optimization issue and an enhanced DE approach is applied to resolve the issue. In [17], proposed an approach to resolve allocating tasks through Ant Colony Optimization (ACO) by adopting fault tolerance and Reinforcement learning (RL) for making fault-tolerant scheduling process, and to accomplish the goal of minimal makespan. Arora et al. [18] presented the method to handle various requests from multi-user to the cloud. It contains two parts: initially, each VM is monitored and the optimum VM with adequate resources to manage the request is carefully chosen and next, the task is assigned to the essential resource. Also, check the load on every VM by continuously observing, and once the load is improved, relocation can be performed to the following VM. In the next section 3, discusses proposed model, followed in section 4 experimental evaluation section, the final section discusses conclusion with future findings.

3 The Proposed Model

This study has designed a new GTO-FTASS technique for scheduling tasks and allocating resources in the CC environment. The GTO-FTASS model has computed a fitness function comprising two parameters such as ETC and failure probability of executing a task. In addition, the presented fault detector can trace the failed tasks or VMs and later schedule healing sub-modules in sequence with a remedial or retrieval scheduling model.

3.1 Overview of GTO

Abdollahzadeh et al. proposed GTO algorithm which depends upon the social intelligence of GTO naturally. Here, the behavior of gorillas is arithmetically modelled by utilizing novel mechanisms for exploitation and exploration stages. Furthermore, it lives under the leader of silverback gorilla who take each troop's decision and hunts for food in a group. This approach considers that the weaker solutions in the population are weaker gorilla in the set. Furthermore, another gorilla seeks to move away to attain optimal solutions for enhancing each gorilla position. Moreover, the exploitation and exploration stages of the GTO is briefly discussed and summarized in the following [19]:

Fig. 1 shows the flowchart of GTO algorithm. In the exploration stage, three distinct models (moves to another gorilla, migration towards unknown position, and migration to a known position) that is represented by Eq (1):

$$GX(t+1) = \begin{cases} (U-L)r_1 + L & if\ r\ and < p(r_2 - D)X_r(t) + S \times V \quad if\ r\ and \geq 0.5 \\ X(t) - S(S(X(t) - GX_r(t)) + r_3(X(t) - GX_r(t))) & if\ r\ and < 0.5 \end{cases} \tag{1}$$

whereas $GX(t+1)$ represents the location vector in the following t iteration, L an U d denotes the lower and upper bounds of the variable, correspondingly. $X(t)$ represents the present vector of gorilla location, X_r indicates gorilla in a group that is arbitrarily selected from the population and $GX_r(t)$ denotes the location of random gorilla. Furthermore, rand, r_1, r_2 and r_3 represent arbitrary parameters from 0 to 1. The parameter p denotes the possibility of choosing the migration to unknown position. Furthermore, the S and V variables denote the silverback motion. The variable D is computed by the following equation:

$$D = N \times \left(1 - \frac{t}{MaxIt}\right) \tag{2}$$

Now N represent the population size and $MaxIt$ indicates the maximal amount of iterations.

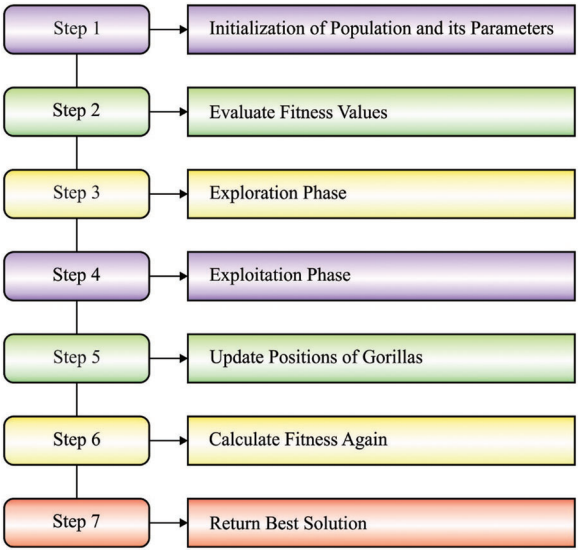

Figure 1: Flowchart of GTO algorithm

At the exploitation stage, two distinct models are employed. They are depending on a comparison among the D value estimated by a parameter W. The gorilla obeys each silverback command to go to different positions for searching food supplies and it can be used only if $D \geq W$ and it is formulated in Eq. (3):

$$GX(t+1) = S \times J(X(t) - X_{silverback}) + X(t) \tag{3}$$

In which $X_{silverback}$ indicates the location vector of silverback. Furthermore, once young gorillas grow up, they compete with male gorillas on the adult female. It can be used only if $D < W$ and also it is expressed in the following equation:

$$GX(i) = X_{silverback} - (X_{silverback} \times R - X(t) \times R) \times T \tag{4}$$

whereas T and R represent coefficient and the impact force of violence level, correspondingly. At last, the fitness function (FF) solution is upgraded by the optimal solution at the end of the exploitation stage.

Algorithm 1: Pseudocode of GTO algorithm

Initializes population size N and the maximal amount of iterations *Maxiter*

Initializes the gorilla population randomly $X_i(i = 1, 2, \cdots, N)$

Evaluate fitness value of each gorilla individual

While $t <$ *Maxiter* do

Upgrade the variable C

Upgrade the variable L

For every gorilla X_i do//Exploration phase

Upgrade the position of the present gorilla

End For

Calculate the fitness value of each gorilla

(Continued)

Algorithm 1: (continued)

Store optimum solutions as silverback $X_{silverback}$
For every gorilla X_i do
If $C \geq W$ then
Upgrade location of present gorilla
Else
Upgrade location of present gorilla
End If
End For
Upgrade the fitness value of each gorilla
Upgrade the global optimal solution $X_{silverback}$
Incerment t
End While
Display global optimal solution $X_{silverback}$ and fitness values

In the following, steps of GTO approach are given:

a) Initializing the population using arbitrary position.
b) Set the variables $MaxIt$, U, L, D, and W.
c) Find the gorilla location.
d) Estimate the FF.
e) Set the optimal solution as the position of the silverback.
f) Upgrade the gorilla location according to D and W values.
g) Show the optimal gorilla location and upgrade the FF until the maximal amount of iterations is obtained.

3.2 Design of GTO-FTASS Model

For deriving the fitness function, the primary objective of the provider is to minimize the execution time, whereas the goal of client is to minimize the cost of retrieving cloud resource by decreasing the make-span time. Thus, the fitness values of GTO-FTASS model are calculated by the fitness function as follows

$$f(C) = \max \left\{ \bigcup_{i=1}^{m} C_i \right\} \tag{5}$$

whereas $C_i^{'}$ denotes the completion of task i. The lower the make-span the higher the efficacy of the model, which means lesser time is taken to implement the process. The expected time of completion (ETC) is described as the implementation time for all the tasks to calculate on a VM attained by the ETC matrix as follows [20]. An ETC matrix associated with n tasks $T = \{T_1, T_2, \ldots, T_n\}$ and m VM signified as $V = \{V_1, V_2, \ldots, V_m\}$ resource

$$\text{ETC} = T \cdot V = \begin{Bmatrix} T_1V_1 & T_1V_2 & \dots & T_1V_m \\ T_2V_1 & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ T_nV_1 & \dots & \dots & T_nV_m \end{Bmatrix} \tag{6}$$

As well, i_i, r_k) is described as the failure probability of performing the task with a trust level (TL) and security demand (SD). The SD represents the security demand for the applications during task submission. The trust mechanism evaluates the VM site dependability, to be accurate, the TL. A task failure method can be defined as a function of the variance between the machine security and task demand. Eq. (7) describes the failure probability about scheduling of task T with a certain SD_j value, to the VM_i with trust value TL_i. TL represents the security assurance for the resource VM, the superior is the TL value the more advanced the VM dependability.

$$P_f(T_i,\ V_k) = \begin{cases} 0 & if\ SD_i \le TL_k \\ 1 - e^{-\alpha\left(SD_j - TL_k\right)}, & if\ SD_i > TL_k \end{cases} \tag{7}$$

In which α indicates the failure coefficient *i.e.*, fraction number.

3.3 Fault Detector

A fault detector is essential to design a fault-tolerance model. Several detection methods have different fields of emphasis on certain parameters, for example, fault coverage, performance, complexity, etc [21–23]. The proposed fault tolerant process is given in Fig. 2. Additionally, VM presented a new facility for detecting failure. The task implementation run-on a VM is examined from remote site and the task failure is distinguished by the abnormal internal operation data, such as the abnormal cycle of system performance call. VM detection is accomplished by implementing a detection mechanism positioned in virtual machine manager (VMM) that judge intermittently the fault condition of VM. The function of the fault detector we presented here tracks the failed tasks or VM and later schedule healing sub-modules consecutively with a healing or recovery LCA scheduling method. The healing model is to direct the resulting healing sub-models to recover the fault based on fault category and intensity. The fault healing can be achieved successively until the task is completely recovered. Fig. 2 shows the process involved in fault tolerant stage.

4 Performance Evaluation

In this section, the experimental validation of the GTO-FTASS model is performed using two scenarios. In the first scenario, a set of 5 cloud users with 5 brokers and 2 data centers are involved. Next, in the second scenario, a set of 10 cloud users with 10 brokers and 5 data centers are involved. The results are investigated under varying numbers of tasks. Tab. 1 provides a comprehensive comparative examination of the GTO-FTASS model with existing models [20] on scenario 1 in terms of makespan (MSN), failure ratio (FRR), and failure slowdown (FSD). The experimental values indicated that the GTO-FTASS model has resulted in an effectual outcome on scenario 1.

Fig. 3 illustrates a brief MSN examination of the GTO-FTASS model with recent models under varying tasks in scenario 1. The figure indicated that GTO-FTASS model has resulted in effectual outcomes with minimal values of MSN over the other methods. For instance, with 10 tasks, the GTO-FTASS model has offered lower MSN of 409 whereas the MTCT, MAXMIN, ACO, NSGA-II, and DCLCA models have obtained higher MSN of 636, 622, 594, 580, and 537 respectively. Besides, with 50 tasks, the GTO-FTASS model has offered lower MSN of 466 whereas the MTCT, MAXMIN, ACO, NSGA-II,

and DCLCA models have obtained higher MSN of 1359, 1118, 948, 806, and 622 respectively. Along with that, with 100 tasks, the GTO-FTASS model has provided reduced MSN of 721 whereas the MTCT, MAXMIN, ACO, NSGA-II, and DCLCA models have resulted in increased MSN of 3075, 2508, 1955, 1530, and 991 respectively.

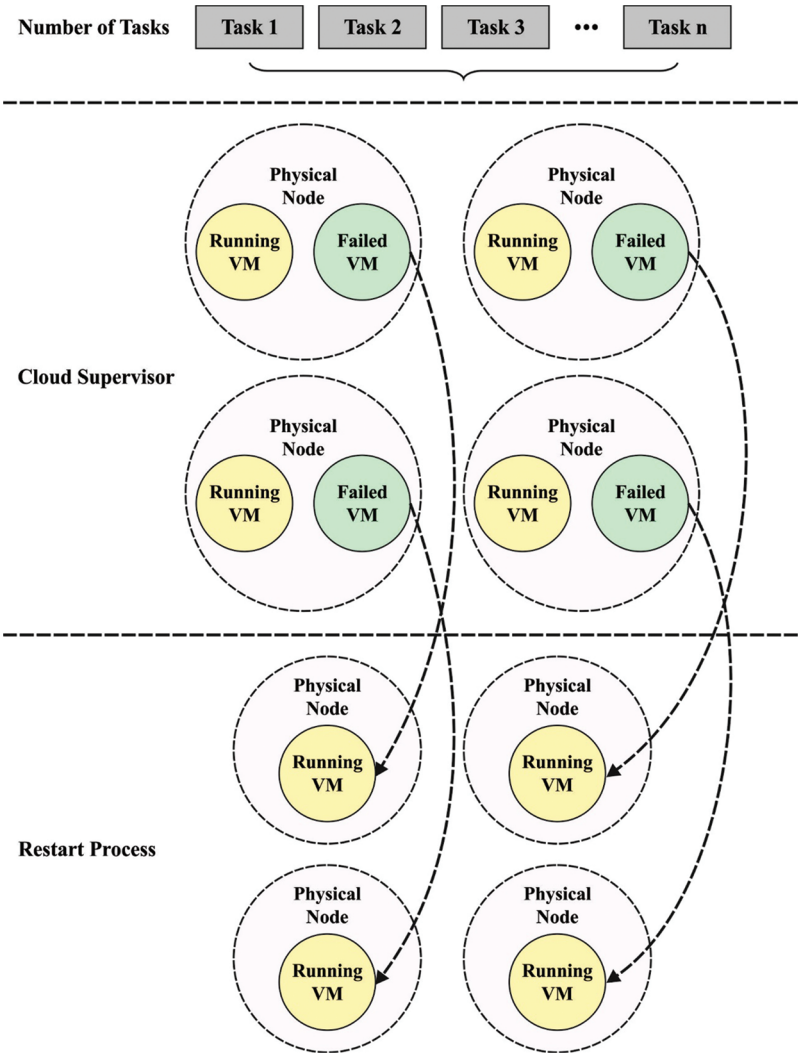

Figure 2: Steps in fault tolerance process

Fig. 4 exemplifies a detailed FRR inspection of the GTO-FTASS model with recent models under variable tasks in scenario 1. The figure designated that GTO-FTASS model has accomplished better results with least values of FRR over the other methods. For instance, with 10 tasks, the GTO-FTASS model has provided minimal FRR of 43.42 whereas the MTCT, MAXMIN, ACO, NSGA-II, and DCLCA models have gained maximum FRR of 63.34, 53.38, 55.51, 50.53, and 50.53 respectively. Meanwhile, with 100 tasks, the GTO-FTASS model has accomplished decreased FRR of 11.41 whereas the MTCT, MAXMIN, ACO, NSGA-II, and DCLCA models have depicted increased FRR of 24.92, 22.08, 23.50, 18.17, and 17.46 respectively. Additionally, with 50 tasks, the GTO-FTASS model has provided lesser FRR of 22.43 whereas the MTCT, MAXMIN, ACO, NSGA-II, and DCLCA models have attained increased FRR of 38.44, 34.53, 34.88, 33.46, and 30.26 respectively.

Table 1: Comparative analysis of GTO-FTASS model under varying tasks in scenario 1

Makespan						
No. of Tasks	MTCT	MAXMIN	ACO	NSGA-II	DCLCA	GTO-FTASS
10	636	622	594	580	537	409
20	665	679	622	665	622	494
30	948	863	835	792	650	551
40	1047	906	877	778	636	480
50	1359	1118	948	806	622	466
60	1487	1274	1317	806	636	409
70	1657	1317	1047	1161	679	494
80	2182	1771	1303	1430	835	594
90	2777	2224	1771	1473	877	622
100	3075	2508	1955	1530	991	721
Failure Ratio						
No. of Tasks	MTCT	MAXMIN	ACO	NSGA-II	DCLCA	GTO-FTASS
10	63.34	53.38	55.51	50.53	50.53	43.42
20	57.29	50.53	44.49	41.29	36.66	32.04
30	50.18	46.98	41.64	36.66	33.82	27.77
40	44.13	41.29	40.93	31.68	29.90	26.70
50	38.44	34.53	34.88	33.46	30.26	22.43
60	33.46	31.33	29.90	29.19	28.48	20.66
70	29.90	29.90	28.84	27.41	28.48	19.95
80	28.84	28.48	28.48	25.64	23.86	16.03
90	27.06	27.77	25.64	20.66	19.59	13.54
100	24.92	22.08	23.50	18.17	17.46	11.41
Failure Slowdown						
No. of Tasks	MTCT	MAXMIN	ACO	NSGA-II	DCLCA	GTO-FTASS
10	1.91	1.47	1.50	1.50	1.08	0.54
20	2.45	2.28	2.03	1.64	1.30	0.86
30	2.84	2.60	2.30	1.81	1.54	1.10
40	3.06	2.82	2.72	2.43	2.13	1.50
50	3.04	2.96	2.87	2.62	2.57	1.76
60	3.82	3.40	3.06	2.94	2.69	1.96
70	4.19	3.97	3.55	2.94	2.65	2.13
80	4.58	4.21	3.82	3.23	2.91	2.33
90	5.31	4.58	3.99	3.55	3.16	2.45
100	6.19	4.55	4.14	3.84	3.53	2.74

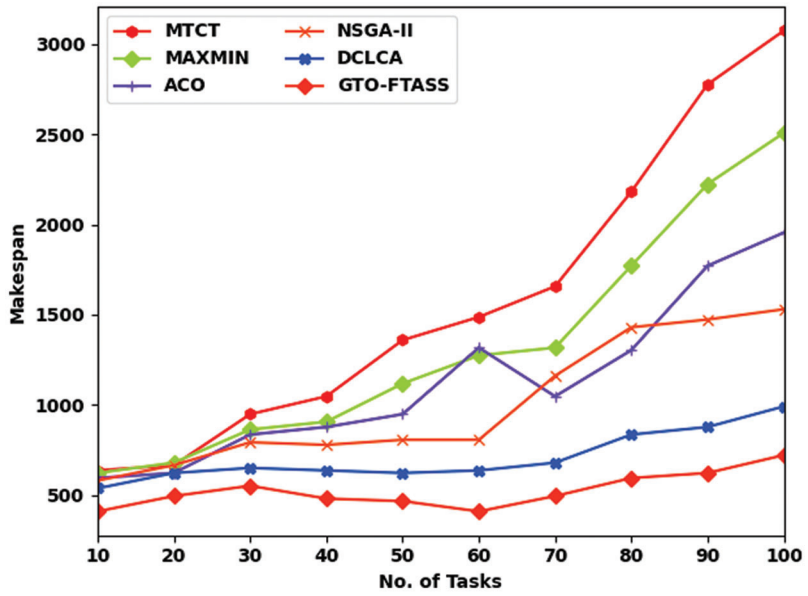

Figure 3: MSN examination of GTO-FTASS model under varying tasks in scenario 1

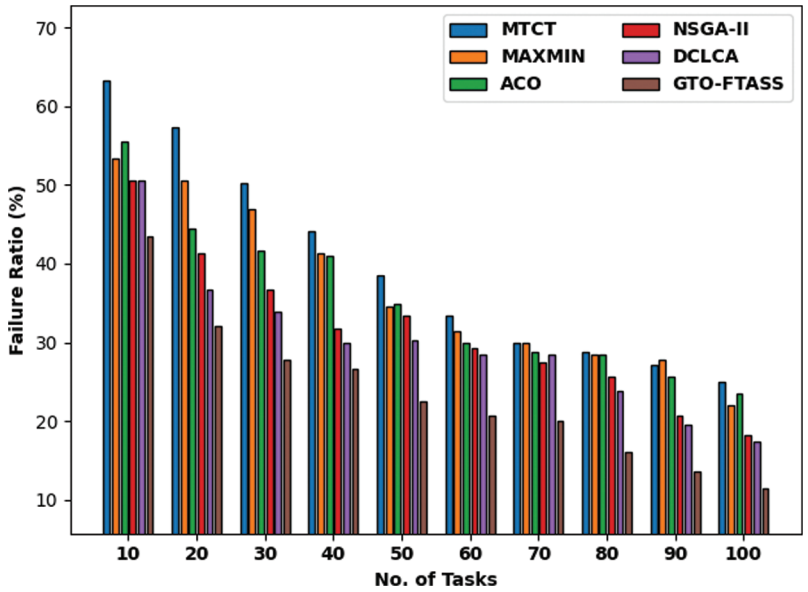

Figure 4: FRR examination of GTO-FTASS model under varying tasks in scenario 1

Fig. 5 demonstrates a comprehensive FSD investigation of the GTO-FTASS model with recent models under changeable tasks in scenario 1. The figure reported that GTO-FTASS model has led to superior performance with lower values of FSD over the other methods. For instance, with 10 tasks, the GTO-FTASS model has accomplished reduced FSD of 0.54 whereas the MTCT, MAXMIN, ACO, NSGA-II, and DCLCA models have demonstrated increased FSD of 1.91, 1.47, 1.50, 1.50, and 1.08 respectively. Besides, with 50 tasks, the GTO-FTASS model has reached minimal FSD of 1.76 whereas the MTCT, MAXMIN, ACO, NSGA-II, and DCLCA models have obtained higher FSD of 3.04, 2.96, 2.87, 2.62, and 2.57 respectively. Along with that, with 100 tasks, the GTO-FTASS model has provided reduced

FSD of 2.74 whereas the MTCT, MAXMIN, ACO, NSGA-II, and DCLCA models have resulted in increased FSD of 6.19, 4.55, 4.14, 3.84, and 3.53 respectively.

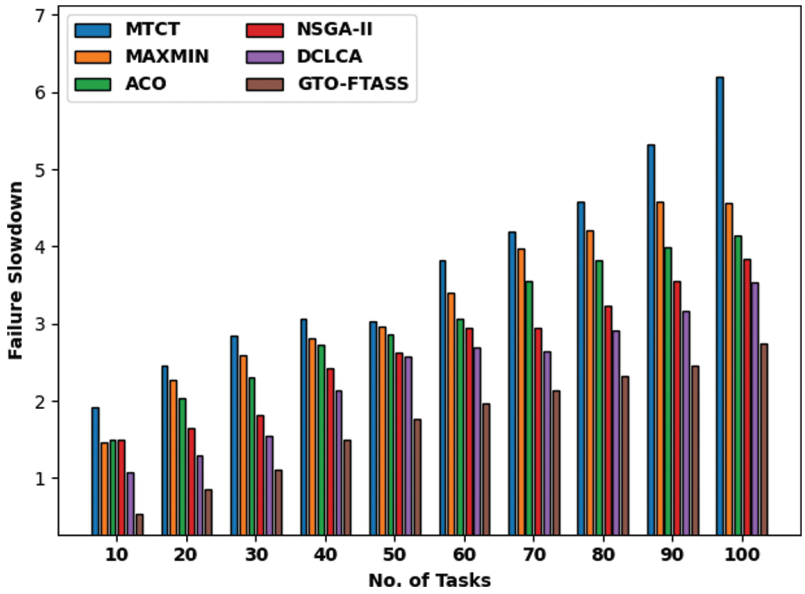

Figure 5: FSD examination of GTO-FTASS model under varying tasks in scenario 1

Tab. 2 offers a detailed comparative study of the GTO-FTASS model with existing models on scenario 2. Fig. 6 illustrates a brief MSN examination of the GTO-FTASS model with recent models under varying tasks in scenario 2. The figure indicated that GTO-FTASS model has resulted in effectual outcomes with minimal values of MSN over the other methods. For instance, with 10 tasks, the GTO-FTASS model has offered lower MSN of 1772 whereas the MTCT, MAXMIN, ACO, NSGA-II, and DCLCA models have obtained higher MSN of 2842, 3043, 2742, 2909, and 2341 respectively. In line with this, with 100 tasks, the GTO-FTASS model has provided reduced MSN of 721 whereas the MTCT, MAXMIN, ACO, NSGA-II, and DCLCA models have resulted in increased MSN of 8828, 6353, 5852, 6253, and 4347 respectively.

Table 2: Comparative analysis of GTO-FTASS model under varying tasks in scenario 2

	Makespan					
No. of Tasks	MTCT	MAXMIN	ACO	NSGA-II	DCLCA	GTO-FTASS
10	2842	3043	2742	2909	2341	1772
20	2876	3311	3879	3210	2107	1806
30	4916	4916	4247	4581	2976	2207
40	5818	5049	4949	4648	3110	2776
50	6353	5250	4949	4882	3712	3244
60	6721	5350	5116	4916	3846	3411
70	6955	5551	5350	5150	3979	3545
80	7724	6019	5283	5484	4247	3812

Table 2 (continued)

Makespan						
No. of Tasks	MTCT	MAXMIN	ACO	NSGA-II	DCLCA	GTO-FTASS
90	8259	6119	5651	5919	4213	3846
100	8828	6353	5852	6253	4347	4046
Failure Ratio						
No. of Tasks	MTCT	MAXMIN	ACO	NSGA-II	DCLCA	GTO-FTASS
10	69.69	65.89	63.82	51.75	53.82	43.82
20	66.58	62.79	61.75	52.10	49.34	39.68
30	60.72	58.31	57.96	49.34	44.85	39.68
40	55.89	53.13	52.79	45.20	41.41	34.85
50	54.86	51.41	50.72	44.51	38.99	31.75
60	51.06	46.92	46.58	38.99	33.47	24.51
70	50.37	45.20	42.44	32.44	31.06	18.64
80	47.96	40.72	37.27	35.20	23.82	12.09
90	41.41	37.27	35.54	29.68	18.99	9.33
100	35.54	30.02	26.23	22.44	11.75	5.19
Failure Slowdown						
No. of Tasks	MTCT	MAXMIN	ACO	NSGA-II	DCLCA	GTO-FTASS
10	3.58	2.51	2.30	1.77	1.48	0.65
20	4.81	4.77	4.73	2.18	1.81	1.52
30	6.13	5.59	5.55	3.41	3.16	2.34
40	6.05	6.09	5.96	5.10	4.73	3.41
50	6.95	6.83	6.75	5.68	5.39	4.24
60	7.61	7.28	7.16	6.29	5.84	4.85
70	8.48	8.06	7.94	7.53	6.66	5.06
80	8.76	8.39	8.27	7.94	7.08	5.22
90	9.38	8.97	8.64	8.06	7.41	5.47
100	9.59	9.22	9.01	8.11	7.78	6.05

Fig. 7 depicts a clear FRR analysis of the GTO-FTASS model with existing approaches under dissimilar tasks in scenario 2. The results portrayed that GTO-FTASS model has reached improved performance with reduced values of FRR over the other methods. For instance, with 10 tasks, the GTO-FTASS model has reached inferior FRR of 43.82 whereas the MTCT, MAXMIN, ACO, NSGA-II, and DCLCA models have displayed enhanced FRR of 69.69, 65.89, 63.82, 51.75, and 53.82 respectively. Eventually, with

100 tasks, the GTO-FTASS model has provided reduced FRR of 5.19 whereas the MTCT, MAXMIN, ACO, NSGA-II, and DCLCA models have resulted in increased FRR of 35.54, 30.02, 26.23, 22.44, and 11.75 respectively.

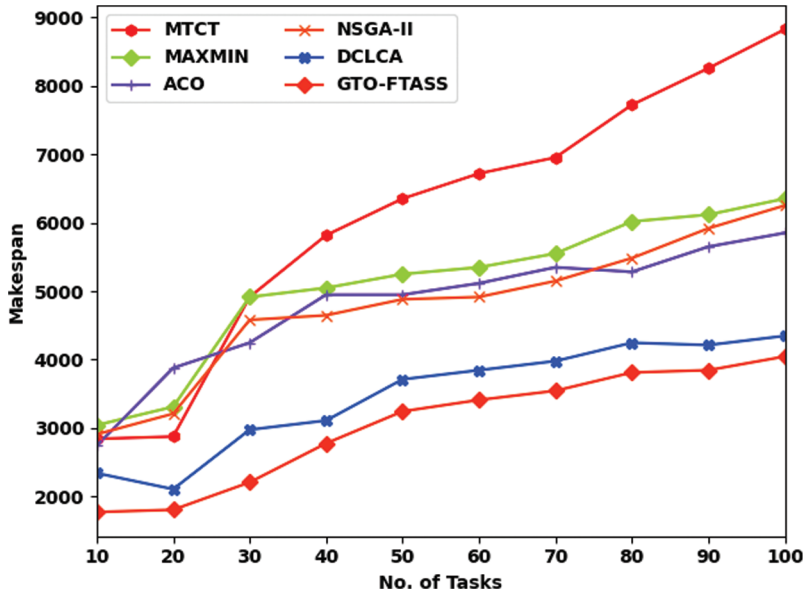

Figure 6: MSN examination of GTO-FTASS model under varying tasks in scenario 2

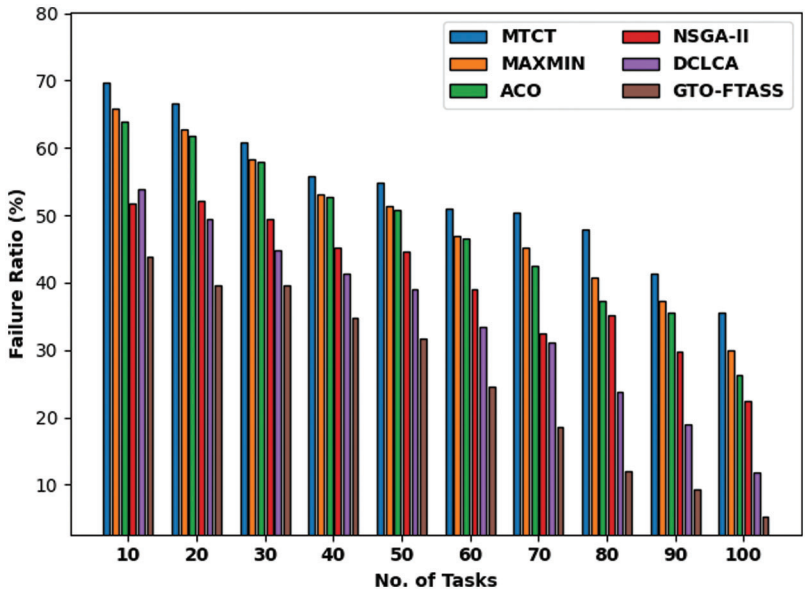

Figure 7: FRR examination of GTO-FTASS model under varying tasks in scenario 1

Fig. 8 explains a brief FSD assessment of the GTO-FTASS model with existing approaches under varying tasks in scenario 2. The experimental outcome revealed that GTO-FTASS model has accomplished better results than the other methods with minimal values of FSD over the other methods. For instance, with 10 tasks, the GTO-FTASS model has provided least FSD of 0.65 whereas the MTCT,

MAXMIN, ACO, NSGA-II, and DCLCA models have gained increased FSD of 3.58, 2.51, 2.30, 1.77, and 1.48 respectively. Similarly, with 100 tasks, the GTO-FTASS model has provided reduced FSD of 6.05 whereas the MTCT, MAXMIN, ACO, NSGA-II, and DCLCA models have resulted in increased FSD of 9.59, 9.22, 9.01, 8.11, and 7.78 respectively.

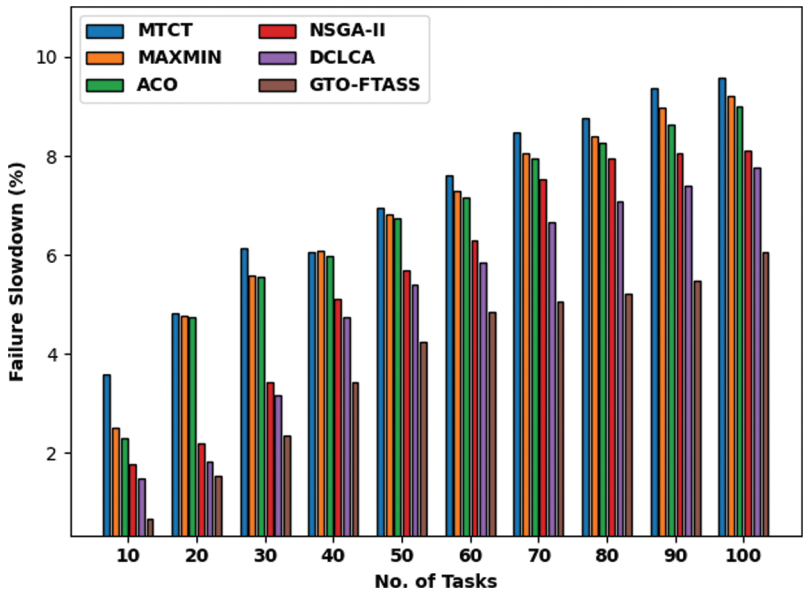

Figure 8: FSD examination of GTO-FTASS model under varying tasks in scenario 1

After examining the above mentioned tables and figures, it is apparent that the GTO-FTASS model has resulted in effectual scheduling performance in the CC environment.

5 Conclusion

This study has designed a new GTO-FTASS technique for scheduling tasks and allocating resources in the CC environment. The GTO-FTASS model has computed a fitness function comprising two parameters such as ETC and failure probability of executing a task. In addition, the presented fault detector can trace the failed tasks or VMs and later schedule the healing sub-module in sequence with a remedial or retrieval scheduling model. The experimental validation of the GTO-FTASS model has been performed and the results are inspected under several aspects. The extensive comparative analysis reported the better outcomes of the GTO-FTASS model over the recent approaches. As a part of future scope, the GTO-FTASS technique can be applied as an effective tool for accomplishing effective scheduling and fault tolerance in CC environment.

Funding Statement: The authors received no specific funding for this study.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] C. Kathpal and R. Garg, "Survey on fault-tolerance-aware scheduling in cloud computing," in *in proc. Int. Conf. on Information and Communication Technology for Competitive Strategies*, Singapore, Springer, pp. 275–283, 2019.

[2] A. Alarifi, F. Abdelsamie and M. Amoon, "A fault-tolerant aware scheduling method for fog-cloud environments," *PloS one*, vol. 14, no. 10, pp. 1–24, 2019.

[3] V. Mohammadian, N. J. Navimipour, M. Hosseinzadeh and A. Darwesh, "Comprehensive and systematic study on the fault tolerance architectures in cloud computing," *Journal of Circuits, Systems and Computers*, vol. 29, no. 15, pp. 2050240–2050252, 2020.

[4] Z. Ahmad, B. Nazir and A. Umer, "A fault-tolerant workflow management system with Quality-of-Service-aware scheduling for scientific workflows in cloud computing," *International Journal of Communication Systems*, vol. 34, no. 1, pp. 1–18, 2021.

[5] M. Khaldi, M. Rebbah, B. Meftah and O. Smail, "Fault tolerance for a scientific workflow system in a cloud computing environment," *International Journal of Computers and Applications*, vol. 42, no. 7, pp. 705–714, 2020.

[6] H. Sun, H. Yu, G. Fan and L. Chen, "QoS-aware task placement with fault-tolerance in the edge-cloud," *IEEE Access*, vol. 8, pp. 77987–78003, 2020.

[7] V. Mohammadian, N. J. Navimipour, M. Hosseinzadeh and A. Darwesh, "Comprehensive and systematic study on the fault tolerance architectures in cloud computing," *Journal of Circuits, Systems and Computers*, vol. 29, no. 15, pp. 2050240, 2020.

[8] X. Xu, R. Mo, F. Dai, W. Lin, S. Wan *et al.,* "Dynamic resource provisioning with fault tolerance for data-intensive meteorological workflows in cloud," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 9, pp. 6172–6181, 2019.

[9] M. R. Thanka, P. Uma Maheswari and E. B. Edwin, "An improved efficient: Artificial bee colony algorithm for security and qos aware scheduling in cloud computing environment," *Cluster Computing*, vol. 22, no. 5, pp. 10905–10913, 2019.

[10] D. Alsadie, "A metaheuristic framework for dynamic virtual machine allocation with optimized task scheduling in cloud data centers," *IEEE Access*, vol. 9, pp. 74218–74233, 2021.

[11] P. Kasu, P. Hamandawana and T. S. Chung, "DLFT: Data and layout aware fault tolerance framework for big data transfer systems," *IEEE Access*, vol. 9, pp. 22939–22954, 2021.

[12] S. Kanwal, Z. Iqbal, F. Al-Turjman, A. Irtaza and M. A. Khan, "Multiphase fault tolerance genetic algorithm for vm and task scheduling in datacenter," *Information Processing & Management*, vol. 58, no. 5, pp. 102676–102688, 2021.

[13] V. Sathiyamoorthi, P. Keerthika, P. Suresh, Z. J. Zhang, A. P. Rao *et al.,* "Adaptive fault tolerant resource allocation scheme for cloud computing environments," *Journal of Organizational and End User Computing (JOEUC)*, vol. 33, no. 5, pp. 135–152, 2021.

[14] W. Jing, C. Zhao, Q. Miao, H. Song and G. Chen, "QoS-DPSO: QoS-aware task scheduling for cloud computing system," *Journal of Network and Systems Management*, vol. 29, no. 1, pp. 1–29, 2021.

[15] M. Alaei, R. Khorsand and M. Ramezanpour, "An adaptive fault detector strategy for scientific workflow scheduling based on improved differential evolution algorithm in cloud," *Applied Soft Computing*, vol. 99, no. 6, pp. 106895–106907, 2021.

[16] L. Karthikeyan, C. Vijayakumaran, S. Chitra and S. Arumugam, "Saldeft: Self-adaptive learning differential evolution based optimal physical machine selection for fault tolerance problem in cloud," *Wireless Personal Communications*, vol. 118, no. 2, pp. 1453–1480, 2021.

[17] J. Nalini and P. M. Khilar, "Reinforced ant colony optimization for fault tolerant task allocation in cloud environments," *Wireless Personal Communications*, vol. 121, no. 4, pp. 2441–2459, 2021.

[18] A. Arora, B. Talwar and S. Bharany, "Reliability aware mechanism to ensure increased fault tolerance using throttle load balancer," in *Proc. 9th Int. Conf. on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, Noida, India, pp. 1–5, 2021.

[19] B. Abdollahzadeh, F. S. Gharehchopogh and S. Mirjalili, "Artificial gorilla troops optimizer: A new nature-inspired metaheuristic algorithm for global optimization problems," *International Journal of Intelligent Systems*, vol. 36, no. 10, pp. 5887–5958, 2021.

[20] S. I. M. Abdulhamid, M. S. Abd Latiff, S. H. H. Madni and M. Abdullahi, "Fault tolerance aware scheduling technique for cloud computing environment using dynamic clustering algorithm," *Neural Computing and Applications*, vol. 29, no. 1, pp. 279–293, 2018.

[21] T. S. Kumar, H. S. Madhusudhan, S. M. F. D. Mustapha, P. Gupta and R. P. Tripathi, "Intelligent fault-tolerant mechanism for data centers of cloud infrastructure," *Mathematical Problems in Engineering*, vol. 2022, no. 2, pp. 1–12, 2022.

[22] R. Saravanakumar, N. Krishnaraj, S. Venkatraman, B. Sivakumar, S. Prasanna *et al.,* "Hierarchical symbolic analysis and particle swarm optimization based fault diagnosis model for rotating machineries with deep neural networks," *Measurement*, vol. 171, no. 108771, pp. 1–13, 2021.

[23] B. Varghese and R. Buyya, "Next generation cloud computing: New trends and research directions," *Future Generation Computer Systems*, vol. 79, no. 6, pp. 849–861, 2018.