An Empirical Investigation of Intelligent Fault-Tolerant Strategy for Cloud Computing System

Kalim Qureshi * and Taiba Yousef Alkhurafi

Department of Information Science, College of Life Sciences, Kuwait University, Kuwait
Email: kalimuddin.qureshi@ku.edu.kw (K.Q.); tayyeba.alkhurafi@grad.ku.edu.kw (T.Y.A.)
*Corresponding author

***Abstract*—Cloud computing is a transformative technology in distributed computing. As a result of the increasing demand for accessing data through the Internet by various endpoints, such as laptops, smartphones, and mobile and wearable devices, cloud services continue to grow and thrive. However, the extensive use of cloud-based services increases the risks of failure, compromising the service's reliability and availability. To manage failures, cloud computing systems must be fault-tolerant. Although there are many techniques for fault tolerance in cloud computing, they lack a holistic solution for multiple layers of the cloud infrastructure. Additionally, as the complexity of the cloud infrastructure increases, there is an urge to provide an intelligent strategy to deal with failures. In this investigation, we introduce an Intelligent Fault-Tolerant Strategy (IFTS) that will respond to cloud failures efficiently. The proposed IFTS provided a solution for different cloud infrastructure layers. The strategy was simulated using the CloudSim simulator, and the main results of our experiments are as follows; host selection process in IFTS provided a better makespan by 11.57% than random selection and by 22.05% than the worst-fit process, with a 0% failure rate. IFTS cloudlet to Virtual Machine (VM) mapping process managed to decrease the failure rate by 28.08% in random mapping and 15.46% in available VM mapping to 5.91%, which is a huge difference and improvement. The results for application-level Fault Tolerant (FT) show that IFTS makespan is improved by 27.52% when using replication and by 20.83% when using checkpointing.**

***Keywords*—cloud computing, Fault Tolerant (FT), host reliability, Virtual Machine (VM) allocation, replication, checkpointing**

I. Introduction

Cloud computing has become an emerging paradigm in information technology and has gained attention in academia and the corporate world. One of the reasons behind the popularity of cloud computing is that it delivers flexible, scalable, and on-demand services without the need for an upfront investment, which can reduce costs in some organizations [1]. In a cloud environment, customers can easily manage the variation in demands by simply scaling the IT resources up or down without the intervention of the Cloud Service Provider (CSP) or a third party [2]. With recent technological advances, such as social networks, artificial intelligence, and big data analysis, the demand for cloud services has increased extensively, thereby increasing the chances of failures. Failures in cloud computing systems are crucial as they might lead to huge financial losses. Therefore, to mitigate the impact of failures, cloud computing systems must be Fault-Tolerant (FT). An efficient FT strategy is vital since it ensures that cloud operations are running as expected and avoids any chances of service outages and downtime. Achieving high availability and reliability is critical to the cloud since any related issues might affect the Service Level Agreement (SLA) between the CSP and users. To maintain high availability and reliability, an effective FT design must be embedded in the cloud architected to allow the continuation of services in the presence of faults.

A. Fault Tolerance Challenges in Cloud Computing

The extensive use of the cloud infrastructure compromises service reliability and availability [3]. Cloud architecture has become more complex and dynamic, which makes it error-prone to failures. Therefore, FT has become a core issue for any cloud-based system and must be critically analyzed to avoid any cause of interruption to business activities and lead to potential financial losses. As mentioned earlier, many solutions have been proposed for fault tolerance in cloud computing, such as failure prediction, checkpointing, replications, and VM migration; however, implementing the technique itself can be quite challenging. Cloud computing systems are highly scalable, and the state of the system varies according to the users' demands. Therefore, FT solutions must be scalable to support the scalability characteristic of the cloud, which creates a burden and raises the chances of faults occurrence. In cloud environments, failures can happen at any time for multiple reasons, such as hardware failure, resource failure, or software failure; thus, providing a holistic solution to address all possible failures is a hard problem. Additionally, implementing FT techniques requires additional resources and utilities, such as extra storage or decision-making agents, which might increase the operational costs and affect the energy consumption level of the cloud infrastructure [4]. Lastly, implementing FT algorithms creates another challenge in managing the trade-off between cloud performance and FT complexity and efficiency.

Manuscript received December 18, 2024; revised March 8, 2025; accepted April 14, 2025; published August 18, 2025.

Preventing the effects of failure in cloud computing systems is an important issue. If the CSP doesn't handle failures effectively, failures may result in service outages and downtime. Many outage incidents have been reported due to ineffective FT mitigations. For example, in 2013, Amazon cloud services were down for 49 minutes. The downtime has caused Amazon online stores a revenue loss of around 5 million dollars. Similarly, Windows Azure, Bluehost, and Yahoo Mail faced service outages in 2013, which affected millions of users [5]. Nowadays, users expect to have the highest availability and uptime that reaches 24/7; thus, any unresolved downtime can be costly and harm the CSP's reputation and credibility. No matter what type of cloud deployment is implemented, public or private, outages and unavailability of cloud services caused by unresolved faults is inevitable.

Cloud computing systems relies on virtualization, where the processing is done through remote computers. The risk of VM failure increases with the extensive use of the cloud and the extra demands of processing tasks in several VMs and hosts. Each host can create multiple VMs and each VM can serve multiple cloudlets. FT techniques must consider failures on both VMs and Physical Machines (PM) to increase the reliability of the cloud. When a VM fails to execute the cloudlet successfully, it creates a failure in delivering the task to the user. This failure will affect the reliability of the VM. Allocating VMs to cloudlets cannot be random; otherwise, the risk of SLA violation increases. On the other hand, failure of the PM may not only result in unavailability of the service, but it may cause permanent loss of data, which is unacceptable. Several algorithms monitor PM health through parameters like Central Processing Unit (CPU) temperature, fan speed, and Randum Access Memory (RAM) utilization. The monitoring process will result in VM migrations if the health of the PM is not acceptable or reaches a certain threshold. Nowadays, technology is changing frequently, especially for software. New software upgrades and installations may require new hardware specifications, so using old hardware may lead to incompatibility issues and result in hardware failures. Therefore, keeping the cloud infrastructure up to date and supported by the new software/hardware upgrades and installation creates a challenge and an open space for failures.

Another concern is that tasks submitted to the clouds are variant and have different characteristics and constraints. For example, critical tasks such as real-time ones have tight time constraints, and if the cloud doesn't deliver them on time, the effect will be dramatic [6]. Other tasks require long processing and analysis; the execution on the VM can last for days to months. Failure to deliver those tasks correctly will impact the SLA. Therefore, considering the characteristics of the tasks can play a major role in supporting the proposed solution for FT.

The FT challenges in cloud computing systems include the following:

- FT in cloud computing systems must be critically analyzed to avoid any cause of interruption to business activities and lead to potential financial losses.
- The cloud infrastructure requires a high level of FT to ensure cloud characteristics are not compromised. The strategy must be scalable, address most possible failure types, and manage the trade-off between performance and reliability.
- Implementing FT techniques requires the usage of additional resources and utilities such as extra storage or decision-making agents, which might increase the operational costs and also might affect the energy consumption level of the cloud infrastructure.
- Keeping cloud computing systems up to date requires constant upgrades to its infrastructure (hardware and software). These upgrades create a new challenge for FT strategies with respect to failure detection and mitigation.
- Cloud computing systems have been widely used recently by different types of users; this means that the types of tasks submitted to the cloud differ. Some tasks, such as real-time ones, require fast responses and are considered time-sensitive, whereas other tasks require long processing. Failing to deliver the tasks creates a huge impact on the SLA.
- FT techniques must consider failures on all cloud infrastructure levels to increase the reliability of the cloud system. This might create some delays and affect the service reliability of the cloud.

B. Problem Statement

Much research has been done to implement FT in the cloud, yet the challenges are still persistent. Handling faults in traditional techniques such as checkpoint/restart and replication are insufficient. To build a robust system, an intelligent FT method must be implemented. After studying the researchers' work, we have noticed that some issues haven't been fully addressed or considered. The following points will summarize the problem statement in this investigation:

- Handling FT in cloud computing systems requires an intelligent strategy rather than traditional techniques.
- Tasks submitted to the cloud are not consistent, and they have different classifications. Selecting the FT strategy based on the submitted task classification can increase the level of FT.
- There will always be trade-offs when implementing FT in the cloud, such as complexity and performance.
- Allocating cloudlets to VMs cannot be random. VM selection must be based on a defined approach; otherwise, the risk of SLA violation and failure increases.

Therefore, providing an FT approach in the cloud is not an easy task. The solution must reflect most FT challenges and trade-offs to increase the reliability of the cloud and its ability to operate under failures without any kind of service degradation.

C. Contribution

Our investigation has the following contribution to the cloud computing field and fault tolerance:

- Proposing an intelligent FT strategy that considers multiple infrastructure levels of cloud computing systems.
- Proposing an FT strategy that considers cloud resources reservation to increase the level of reliability of the cloud.
- Proposing an enhanced VM allocation policy based on infrastructure resources to enhance the process of VM allocation to the appropriate host, which increases the reliability of the cloud in the presence of failure.
- Identifying the appropriate VM to allocate the tasks based on their classification and resources needed.
- Proposing smart decision-making to schedule the tasks based on their priority and by using multiple reactive techniques at the application level of the cloud.

II. Related Work

In this section, we present a literature survey in the context of our work. Checkpointing is one of the most commonly used fault tolerance techniques in the cloud environment. Abdulhamid and Abd Latif [7] proposed a checkpointed league championship algorithm. They have used checkpointing to save the system's state periodically; once the task fails, it will be migrated to an underloaded VMs and will start from the last saved checkpoint instead of starting from the beginning. To schedule the failed tasks, a league championship algorithm is used. Ammon *et al.* [8] proposed a solution to enhance the overhead generated by checkpointing when failures occur. Their method uses flexible and varying checkpoints rather than fixed ones. Each VM has a different checkpoint interval based on the failure percentage of the server that carries the VM. Therefore, if the failure percentage is high, checkpoint intervals will be shorter and more frequent. Similarly, Souza *et al.* [9] proposed a hybrid technique that switches between periodic checkpointing (fixed intervals) and on-demand checkpointing. On-demand checkpointing uses an active/active checkpointing technique, which means that when an application starts, two instances will be run on different VMs if their responses are different, they will be synchronized through a checkpoint.

Table I shows an overall summary of the researchers' studies of fault tolerance in cloud computing and provides a number of solutions to increase the reliability of the cloud. The table also shows the shortage and limitations of the previous work.

We developed our proposed strategy in response to the identified limitations. We tried to fill in the gaps of the previously discussed work to design a robust model with a higher FT level that considers the classification of the resources (PM and VM), the nature of tasks submitted to the cloud, and the reliability of the main cloud infrastructure.

TABLE I. Related Work Summary

Ref# year / FT technique	Author contribution	Limitation
[7], 2017 / Hybrid (replication resubmission)	Used checkpointing to save the system's state and when the task fails, it is migrated to the underloaded VM.	They did not consider the VM or PM reliability when task was migrated.
[8], 2019 / Checkpointing	They have link checkpoint intervals with the failure rate of the VM; the higher the failure rate is, the shorter the checkpoint period is set.	They did not discuss what is the approach used when the fault occurs and how VMs are assigned to host.
[9], 2018 / Checkpointing.	They proposed a hybrid technique that switches between periodic checkpointing (fixed intervals) and on-demand checkpointing.	Using one reactive technique to achieve FT might be insufficient for critical tasks such as real-time tasks.
[10], 2020 / VM migration	They suggested monitoring the cloud's vital parameters, which are CPU, fan speed, CPU temperature, and RAM. The migration decision is based on reaching an SLA-defined threshold for each parameter.	They did not identify the properties of the host that the VM will migrate to. Also, although they are monitoring multiple parameters, their experiment picked one parameter CPU to take the decision on VM migration.
[11], 2018 / Replication	The users can choose the level of FT based on a performance/cost trade-off. They have classified the tasks based on their deadline and the user's budget. The more the user pays, the more the FT level and host reliability they will receive.	Budget classification can be inconvenient. They have used a replication strategy for all task categories.
[12], 2017 / Checkpointing	Schedule workflows to meet their soft deadline by dividing the task's deadline into multiple sub deadlines. Initially, the algorithm decides whether to use replication or task resubmission based on deadlines and available resources.	VM or host reliability is ignored.
[13], 2017 / Hybrid (replication resubmission)	Tasks are replicated in all available VMs. If all the VMs failed to execute the task, the reliability factor of the VM will be decreased, and the task will be resubmitted to the VM with the highest reliability factor.	Replicating tasks to all VMs consumes many resources if not justified. The authors did not consider the reliability of the PM.
[14], 2016 / VM migration	They have considered CPU temperature the main parameter to detect a deteriorating physical machine; thus, the VM is migrated proactively to another machine when the CPU temperature reaches a predefined degree.	There are a lot of parameters that cause faults in PM, such as fan speed, CPU utilization, and memory. Depending on CPU temperature only might not give an accurate result to migrate the VM.
[15], 2015 / Replication	They used the PB model, where each task has a primary and backup copy allocated to different hosts.	The VM is selected based on the availability and the task finish time, which doesn't provide a robust FT mechanism.

[16], 2019 / checkpointing replication	Tasks are submitted to primary VMs and the replicated VMs with their status are updated. In case the primary VM fails, checkpointing is used to complete the execution of the tasks in the backup VMs. To increase the level of FT, they have to ensure that tasks and their replicas should not be assigned to the same PM.	Tasks are not classified; also, VM or host reliability is ignored.
[17], 2018 / Hybrid checkpointing replication	Tasks are classified as computing-intensive, data-intensive, or balanced tasks. Each task is duplicated and has a primary and backup copy. Backup copies are checkpointed to determine their execution time. The tasks are then assigned to VMs based on their availability and bandwidth.	They did not use VM or host reliability or failure rate for more FT level.

III. Proposed Strategy

In this section, we present our proposed strategy to increase the FT level of the cloud infrastructure. It explains our vision and idea to fill in some gaps in the literature and provide a new framework that is more reliable and addresses multiple levels in the cloud infrastructure. The full system design is presented, and each step is clarified.

A. Intelligent Fault-Tolerance Strategy (IFTS)

As stated in previous sections, cloud computing systems face many challenges and obstacles. FT is one of the main issues that must be addressed when using cloud computing systems. The aim of FT is to increase the system's reliability in the presence of faults, such that the SLA between the cloud user and CSP is not compromised. Therefore, after surveying the literature and analyzing other researchers' limitations, we have developed a new strategy combining some FT techniques from the literature with our innovative features, as illustrated in Table II.

TABLE II. Features of IFTS

Adapted/ Added Features		Features	Source
Adapted Features	1.	Hybrid application-level FT	[7]
	2.	Resource migration	[10]
	3.	Resource classification	[12, 13]
	4.	VM/host reliability	[14]
Introduced Features	5.	Resources Reservation	Added
	6.	Enhanced tasks/VM mapping	Added
	7.	Smart decision on application FT	Added

The innovative features for IFTS are added based on our analysis of the limitations of other researchers' work as shown in Table II. Those features are as follows:

- Resources reservation: Since IFTS uses resource migration, reserving some resources and dedicating them for FT will speed up the migration process, thus minimizing the migration cost. For example, when the VM fails, the cloud must find a host to migrate the failed VM. Searching for a host in the reserved resources list will guarantee a faster allocation rather than searching for all the host lists that might be fully utilized and cannot place any more VMs on them.
- Resources classification: This feature is derived from the variety of the task types submitted to the cloud. Tasks can be classified by their required resources; they can be I/O-intensive, memory-intensive, or CPU-intensive. Categorizing the resources (host and VM) with the same task types and using this classification allocation and mapping can increase the performance and response time of the cloud.
- Enhanced tasks/VM mapping: One of the major processes in cloud computing is mapping the tasks to appropriate VMs. Many ways were proposed in the literature; however, in the model, we have defined increased the rate of successful mapping by defining multiple cases and scenarios.
- Smart decision on application FT: Handling FT in cloud environments cannot be fulfilled using basic techniques. Adding smart decision-making to its layers can increase the reliability of the cloud. Our proposed strategy uses two of the most used basic FT techniques in literature, replication and checkpointing, and adds a smart decision agent that decides among them based on the task's priority.

B. IFTS Model and Design

The proposed strategy is designed to include multiple levels of the cloud infrastructure. It depends on the host and VM failure rate to allocate the resources. It also considers that tasks submitted to the cloud have different properties and classifications. To simplify the idea, Fig. 1 presents an abstract diagram of the IFTS; each step will be thoroughly explained.

C. Host Selection and VM Creation

In the cloud environment, each data center has a predefined number of PMs (hosts). When the data center is powered on and hosts are available, hosts must be initialized according to three settings. The first setting is the reservation of the resources. The CSP must define a percentage of the number of hosts that will be reserved for FT in case migration is needed. Choosing the right percentage can be tricky and might affect the overall performance of the cloud. For example, choosing a high percentage means that the number of available hosts used for normal operations will decrease, affecting the VM allocating time due to unavailable resources. On the other hand, a low percentage may affect the failure rate of the cloud in the presence of a fault. In this investigation, we have reserved 20% of the resources for FT. The second setting is to eliminate hosts with high failure rates to increase the reliability of the available resources.

Similarly, choosing the accepted failure rate depends on the CSPs and how far they can be strict about their FT policy. The third setting is to categorize the hosts according to their resources. Hosts are divided into three categories: CPU-intensive, memory-intensive, and I/O-intensive.

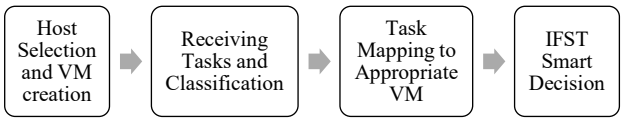

Fig. 1. Abstract diagram of IFTS.

After initializing the hosts and categorizing them, the VM creation process is executed. The CSP has a defined number of VM to be created on the hosts; we have added a new parameter to be set by the CSP, which is the type of the VM. When creating the VM, the main function here is to find a suitable host to place the VM. Algorithm 1 (See Appendix A) shows the steps of this process. First, all hosts will be sorted ascending according to their resources to ensure that VMs will be mapped according to their requirements with a best-fit scenario. Then for each host, the host's type and acceptant failure rate are checked. After that finding, we will iterate through the host list to map the required resources of the VM with the available ones in the host list.

D. Receiving Tasks and Classification

This step is mainly about classifying the submitted tasks to the cloud. When receiving the tasks, the Task Classifier will analyze their requirements and categorize them into their intensive category (CPU, memory, or I/O). An example of how tasks can be classified is provided in Table III.

TABLE III. EXAMPLE OF TASK CLASSIFICATION

Category	Task requirements
CPU-intensive	mips > 1000 RAM = 252 IO = 5
Memory-intensive	mips = 500; ram >384; IO=5
I/O-intensive	mips = 100 io > 20

E. VM Allocation to Appropriate VM

Selecting the appropriate VM to process the cloudlet must be done carefully. This is why we have to define what is the appropriate VM. The mapping process has been implemented by various techniques as follows:

- Mapping the tasks to the available VM.
- Mapping the tasks randomly.
- Mapping the tasks to the most reliable VM.
- Mapping the tasks to the most powerful VM.
- Mapping tasks to best-fit VM.

The effect of each technique is summarized in Table IV as shown below.

TABLE IV. VM ASSIGNMENTS

VM Assignment	Definition	Affect
Available VM	Select the available VM	Selecting the available VM does not mean that it can process the request. More conditions must be checked, such as free resources and reliability for better performance.
Random VM	Select a random VM	Random selection doesn't have a stable performance and is not considered reliable.
Most reliable VM	Select VM with the lowest failure rate	Depending on the failure rate only without considering the task's requirement can result in unfair assignments and tasks that require high reliability, such as real-time tasks might be affected.
Worst-fit VM	Select VM with the highest resources	This assignment has bad resource utilization and increases the execution time of the cloud.
Best-fit VM	Selecting the VM with resources that are close to the task's requirement	This assignment has proper resource utilization.

In IFTS, the process of mapping the cloudlets to the appropriate VM will go through different cases and will execute the steps in Algorithm 2 (see Appendix B). The Algorithm 2 will go through the cloudlets and find the suitable VM to process the request. First, the VMs will be sorted in ascending order to speed up the mapping process when iterating through the list. Then, the failure rate and type of each VM in the list will be checked. After that, the Algorithm 2 starts the mapping process by checking three different cases:

1. Tasks are mapped to the VM of the same category and to the one that has the required resources to execute the request.
2. If VMs of the same category are unavailable, then create another VM of the same category on the reserved resources list.
3. If VMs of the same category are unavailable and gtwe cannot create another VM with the same category, then find a VM with another category that can process the task. The comparison of IFT features with other researchers' work is summarized in Table V.

If the tasks are mapped to a VM, then the tasks will start processing; otherwise, the task will be moved to the task queue and wait for a VM to be available.

TABLE V. COMPARISON OF IFTS FEATURES WITH OTHER RESEARCHERS' WORK

FT Feature / Ref#	Resource classification	VM Reliability	PM Reliability	Resource reservation	Application-level FT		
					Replication	Checkpoint	Smart decision
[10]			✓				
[11]		✓					
[12]	✓		✓				
[13]					✓	✓	
[14]	✓				✓		
[15]						✓	
[16]					✓	✓	
[17]		✓			✓		
IFTS [18]	✓	✓	✓	✓	✓	✓	✓

F. IFTS Smart Decision

All allocation processes are completed at this point, and it is time to start processing the task. As discussed before, tasks have different natures and priorities. Therefore, we have included the task's priority in our application-level FT. Our strategy uses two main reactive FT techniques: replication and checkpointing. Each policy has its pros and cons. Checkpointing implementation does not require many resources, which minimizes its cost and improves cloud performance; however, checkpoint management and rolling-back process create an overhead. Replication has the advantage of parallel execution, which speeds up the delivery of the tasks and has higher performance, but the implementation cost is high. The strategy balances between the two techniques to benefit from their advantages in our favor.

The implementation of the proposed IFTS Smart Decision requires the addition of new components to the cloud architecture. SmartAgent, AppFT Manager. The SmartAgent analyzes the submitted tasks and decides on the application-level FT based on the task's priority. Our proposed model uses replication for tasks with high priority (greater than 0.6) and checkpointing for tasks with low priority. The AppFT manager keeps track of the required information to manage FT on the application level.

The algorithm of the IFTS decision and FT selection is presented in Algorithm 3 (See Appendix C), and its detailed description works as follows:

1. Tasks are placed on their appropriate VMs.
2. The SmartAgent is notified about the initiation of the task's executing process.
3. The SmartAgent will check the task priority to decide what is the used FT policy (replication or checkpointing).
4. The SmartAgent will update the AppFT manager about the decision.
5. The AppFT manager will take the following actions:
   1) In case replication is used, the AppFT manager will create a replica of the task and submit it to the reserved resources list to be executed.
   2) In case checkpointing is used, the AppFT manager creates a periodic checkpoint of the task.
6. When there is a failure at the application level. The AppFT is notified by the broker of the failed task to start the FT recovery process depending on the selected FT policy of the failed task.

G. Comparison of IFTS Features with Other Researchers' Work

To provide an insight into our proposed strategy features, we have listed a comparison of the selected papers in literature with our strategy in terms of the FT techniques related to this investigation in Table IV.

IV. EXPERIMENTAL SETUP AND RESULTS

In this section, we validate our proposed strategy by running multiple tests. IFTS is a strategy that provides an enhanced FT for multiple levels in the cloud. It has four processes illustrated in the abstract diagram as shown in Fig. 1. In this section, we validate three major processes shown in Table VI. Each process is validated by running multiple tests to show its behavior using different setups and scenarios.

TABLE VI. SUMMARY OF IFTS PROCESSES AND ALGORITHMS

Process	Algorithm
Host Selection and VM Creation	Algorithm 1
VM allocation to Appropriate VM	Algorithm 2
IFTS Smart Decision on Application Level	Algorithm 3

A. Experimental Setup

The experiment is performed using the CloudSim simulator. The experiment was run on an Intel Core i7 machine having configuration: 2.6 GHz 6-Core with 16 GB RAM, running macOS Monterary operating system and JAVA Eclipse IDE 4.26 and JDK 14. We have used two evaluation matrices for this experiment, the makespan and average failure rate, as defined in Table VII.

TABLE VII. EVALUATION METRICS

Metric	Definition	Equation	
Make-span	The time when all cloudlets are finished and submitted to the user. In our case, we are using time-shared policy, so it is the average execution time required to process all the cloudlets	$\frac{1}{n}\sum_{i=1}^{n} ti$ where: (ti) is the completion time of the task. (n) number of tasks.	(1)
Failure rate	The stability of the cloud	$\frac{Number\ of\ failed\ Tasks}{Total\ tasks\ submitted}$	(2)

B. Validation of the Host Selection and VM Creation Process

This test aims to validate the host selection and VM creation process of the proposed strategy. Our process is compared with the following host selection processes from the literature:

- Random host selection with fault injection (RND): The host is selected randomly.
- Worst-fit host selection with fault injection (WF): Select the VM with the highest resources.

Best-fit host selection without fault injection (BF): Selecting the host with resources that are close to the VM's requirement without injecting faults into the cloud. This process has been selected to compare the performance of IFTS with the most reliable scenario when no faults disturb the cloud function. The host was configured as shown in Table VIII.

TABLE VIII. HOST CONFIGURATION

Hosts #	PEs #	MIPs	Bandwidth	RAM	Storage
20	10–8	1000–2000	10000–20000	2048 MB	1,000,000

The number of VMs used depends on the scenario that will be explained below with the following configurations, as shown in Table IX.

TABLE IX. VM CONFIGURATION

#PEs	MIPs	Bandwidth	RAM	Storage
2–6	1000	1000–1500	512–1024 MB	100,000

There are three different scenarios used in this test, the configuration of each scenario is provided below.

1) Scenario 1 (SC1): Varying number of cloudlets. In this scenario, the number of cloudlets is set from 200 to 1000, while all other parameters will keep the same configuration, as shown in Table X. This scenario will define how the processes react to the increased demand for the cloud.

TABLE X. SCENARIO 1 CONFIGURATION

#Cloudlets	#VMs	Cloudlet Size	#PEs
200–1000	40	100000000 MIP	2

2) Scenario 2 (SC2): Varying number of VMs. In this scenario, the number of VMs is set from 10 to 50. The purpose of this scenario is to show the behavior when creating a limited number of resources. The configuration is shown in Table XI.

TABLE XI. SCENARIO 2 CONFIGURATION

#Cloudlets	#VMs	Cloudlet Size	#Pes
500	10–50	100000000 MIP	2

Scenario 3 (SC3): Varying Cloudlets Sizes. In this scenario, the size of the cloudlet is dynamic. It will show the effect of the cloudlet's length of the allocation process. The configuration is shown in Table XII.

TABLE XII. SCENARIO 3 CONFIGURATION

#Cloudlets	#VMs	Cloudlet Size	#Pes
500	40	1100000000–1900000000 MIP	2

Another parameter that needs to be set in the host selection process is the acceptance failure rate used. The acceptance failure rate is set to 80%. Additionally, a fault is generated on a random resource at a random time for 1 minute.

C. Results and Analysis of the Host Selection and VM Creation Process

The results of the above configurations are presented in below tables for each scenario. Each test was executed four times, and the values shown are the average of each simulation. Table XI shows the performance comparison in terms of makespan and failure rate for SC1.

SC1 shows that IFTS has a close performance to the makespan of the cloud when no faults are injected (BF), and this is due to the accepted failure rate value, which eliminates the resources with a high failure rate and due to the resource reservation for FT. It also succeeded in delivering all tasks successfully. Table XIII shows the results of executing SC2.

The results show that IFTS has a better makespan and failure rate despite limited resources. The last scenario results, SC3, are shown in Table XIV.

SC3 shows the effect of the cloudlet's size. Having a task that requires long processing means that more resources are required. Although it is observed that there is an increase in the makespan in some cases, we can see that IFTS has delivered all cloudlets with a 0% failure rate.

TABLE XIII. MAKESPAN / FAILURE RATE COMPARISON RESULTS FOR SC1

#	Makespan in MS				Failure Rate %	
Clou-dlets	BF	RND	WF	IFTS	RND	WF
200	55.56	69.44	72.14	61.11	2.01	3.5
300	83.33	111.11	138.89	93.21	3.375	5.75
400	110.9	140.89	176.26	138.89	5.625	7.75
500	180.56	222.36	250.14	188.89	7.75	9.5
600	208.34	277.92	305.69	222.36	9.01	10.75
700	250	361.25	389.03	290.14	10.125	11.75
800	277.78	416.67	444.44	305.69	11.25	12.125
900	319.46	661.53	689.31	361.25	11.75	12.5
1000	347.22	798.01	828.06	416.67	12.125	13
Average	203.683	339.908	365.995	230.912	8.11	9.63

TABLE XIV. MAKESPAN/FAILURE RATE COMPARISON RESULTS FOR SC2

#VMs	Makespan in MS				Failure Rate %	
	BF	RND	WF	IFTS	RND	WF
50	61.11	80.44	90.54	70.11	2.56	7.78
40	69.44	107.22	207.6	80.56	4.09	11.75
30	97.22	325	481.68	210.83	7.89	11.75
20	138.89	169.44	200	140.89	9.89	15.125
10	277.78	644.44	731.58	416.67	14.45	20.88
Average	128.888	265.308	342.28	183.812	7.78	13.46

TABLE XV. MAKESPAN/FAILURE RATE COMPARISON RESULTS FOR SC2

#VMs	Makespan in MS				Failure Rate %	
	BF	RND	WF	IFTS	RND	WF
50	61.11	80.44	90.54	70.11	2.56	7.78
40	69.44	107.22	207.6	80.56	4.09	11.75
30	97.22	325	481.68	210.83	7.89	11.75
20	138.89	169.44	200	140.89	9.89	15.125
10	277.78	644.44	731.58	416.67	14.45	20.88
Average	128.888	265.308	342.28	183.812	7.78	13.46

To give an overview of the performance of all selected processes, Table XV presents the summary of all scenarios' results. To calculate the improvement of IFTS, we have calculated the IFTS makespan efficiency compared with the selected processes using the following equation:

IFTS makespan efficiency=

$$\frac{Makespan\ (p) - Makespan\ (IFTS)}{Makespan\ (p)} \times 100 \quad (3)$$

where $Makespan\ (p)$ is the makespan of the process to be compared with.

We summarize the analysis of the above results in the following points:

1. Both BF and IFTS provided a 0% failure rate in all scenarios, which means that all submitted cloudlets have been returned successfully; thus, we have removed their results from Tables XI–XIV.
2. We have observed that IFTS has a close performance to the makespan of the cloud when no faults are injected (BF), and this is due to the accepted failure rate value, which eliminates the resources with a high failure rate and due to the resources reservation for FT.
3. When comparing IFTS with BF, the increase in makespan in IFTS comes from the effect of the fault injection process. Resources will operate in degraded performance, increasing the IFTS makespan.
4. Results prove that considering the failure rate in the host selection and VM creation process has a huge impact on cloud reliability.
5. SC2 results in Table XII show that creating a limited number of VMs affects the makespan, as the cloudlets will wait for the resources to be available to start processing.
6. After analyzing SC3 results in Table XIII, we observed that the IFTS makespan is increased after setting the cloudlet size to 1400000000 MIP, and it looks like the random selection will work better in some cases. However, the failure rate for IFTS is 0%, which lets us consider the increase in makespan as a minor cost of implementing FT.

Fig. 2 and Table XVI show the efficiency of IFTS. Implementing the strategy has improved performance compared with the selected processes.

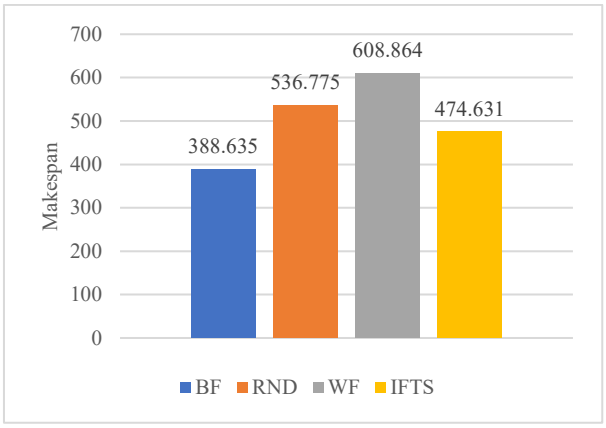

Fig. 2. Comparison of Makespan average for all scenarios and host selection processes.

TABLE XVI. SUMMARY OF MEASURED DATA RESULTS FOR ALL SCENARIOS

Scenario	AVRG Makespan in MS				IFTS Makespan Efficiency in percentage (%)			AVRG Failure rate in percentage (%)	
	BF	RND	WF	IFTS	BF	RND	WF	RND	WF
SC1	203.683	339.908	365.995	230.912	−13.36	32.06	36.90	8.113	9.625
SC2	128.888	265.308	342.28	183.812	−42.61	30.71	46.29	7.776	13.457
SC3	833.333	1005.109	1118.318	1009.17	−21.10	−0.404	9.76	23.281	28.189
Average	388.635	536.775	608.864	474.631	−22.12	11.57	22.05	13.06	17.09

D. Discussion on the Host Selection and VM Creation Process Results

The result of the host selection process test shows that as the number of cloudlets increases, the makespan increases, which is normal behavior. The results for both random and worst-fit techniques show the effect of allocating resources without considering failure rate and appropriate mapping. Additionally, creating a limited number of VMs affects the makespan, as the cloudlets will wait for the resources to be available to start processing.

When other techniques show a good performance, such as random selection, this is because when the cloudlet fails, its makespan calculation will be set to the point of failure. Therefore, having a short makespan without comparing it with the failure rate will not present the full insight into the cloud performance. This test has resulted in the following findings:

- The host selection process in IFTS provided a better makespan by 11.57% than random selection and by 22.05% than the worst-fit process.
- The cost of implementing FT in the makespan of IFTS compared with the best-fit process is 22.12%. However, the results show a noticeable improvement in the IFTS failure rate.

E. Validation of the VM allocation to Appropriate VM Process

This test is conducted to validate the results and compared it with the following processes from the literature:

- Random VM Mapping (RND-VM): When the cloudlets are received, the VM is selected randomly.
- Available VM Mapping (AVL-VM): The mapping will be upon the available VM without checking the category of the VM.

As explained earlier, the algorithm for selecting the appropriate VM undergoes three cases:

1. IFTS CASE 1: Cloudlets will process on the same VM category.
2. IFTS CASE 2: If the VM of the same category is not available, then a VM of the same category will be created on the reserved resources to process the request.
3. IFTS CASE 3: If the VM of the same category cannot be created, then the cloudlet will start processing on a VM of a different category that has sufficient resources.

To analyze the performance of the proposed mapping process, we first ran the test for CASE 1 and compared it with the other processes. Then we expanded the test to include CASE 1 and CASE 2, and the same was done for CASE 3. Hence, by the end of the test, all of the cases are implemented to show the improvements in the makespan and failure rate as we keep building on the algorithm. The cloud was configured as shown in Table XVII.

TABLE XVII. CLOUD CONFIGURATION

Configuration Component Name	Value
Number of Hosts	10
Number of Datacenters	1
Host PEs	8
MIP for Each Core	3000
RAM	2048 MB
Storage	100000 MB
Bandwidth	2000MB
VM Count	30 (Each host has 1 CPU-intensive, 1 memory-intensive, and 1 I/O-intensive)
Number of Cloudlets	30–100, different types

F. Results and Analysis of the VM allocation to Appropriate VM Process

The result of the above configuration is shown in Table XVIII. It shows the behavior of the VM allocation process to the VM in all defined cases with the selected processes.

Our analysis of the above results is summarized as follows:

1. The proposed mapping process performed better than other mapping processes in the makespan and failure rate.
2. Mapping cloudlets to the same VM category (IFTS CASE1) provides a better makespan than AVL-VM by 12.047% and better than RND-VM by 18.14%.
3. When analyzing the failure rate comparison results, we can see that although IFTS did not manage to deliver all the submitted cloudlets successfully, it provided a better failure rate result.
4. After implementing IFTS CASE2, there is an increase in IFTS makespan compared to the results in CASE 1. For example, the makespan in CASE1 for processing 100 cloudlets using IFTS is 802.6 msec, while in CASE 2, it is increased to 960.01 sec. This is due to the time required to create another VM of the same type of cloudlets on a suitable host. With this increase in the makespan comes the advantage of reducing the failure rate from 30% to 22.5%, which is considered an overall algorithm improvement.
5. After implementing IFTS CASE3, the full algorithm of mapping the cloudlets to the appropriate VM is executed. The results have also shown an improvement in delivering the cloudlets successfully by decreasing the failure rate with an increase in the makespan, as illustrated in Figs. 3 and 4.

TABLE XVIII. MEASURED RESULTS OF THE MAKESPAN AND FAILURE RATE FOR IFTS AND OTHER SELECTED PROCESSES

Measured Value	# Cloudlets	RND-VM	AVL-VM	IFTS CASE1	IFTS CASE2	IFTS CASE3
Makespan in MS	30	710.1	650.7	598.85	598.85	598.85
	50	758.85	707.6	601.1	650.2	720.4
	80	882.6	861.35	741.35	870.35	980.2
	100	1000.35	900.1	802.6	960.01	1090.8
	Average	**837.975**	**779.9375**	**685.975**	**769.853**	**847.563**
Failure Rate %	30	5.83	3.33	0	0	0
	50	11.50	6.5	4	3	2.5
	80	33.75	11.25	8.75	6.25	3.37
	100	61.25	40.75	30	22.5	17.75
	Average	**28.08**	**15.46**	**10.69**	**7.94**	**5.91**

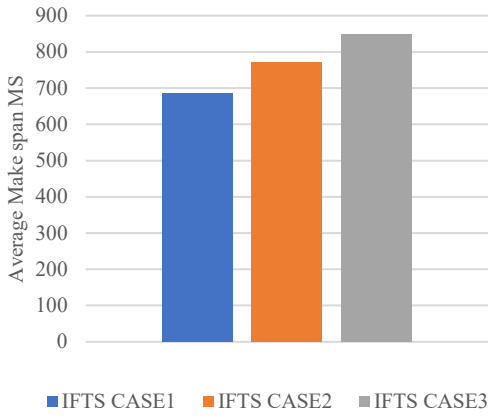

Fig. 3. Makespan increase in IFTS.

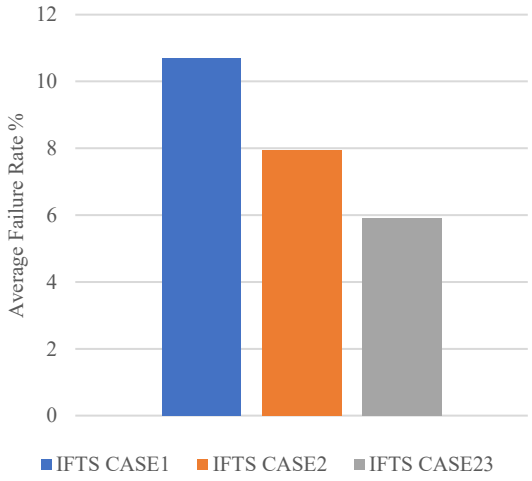

Fig. 4. Failure rate improvement in IFTS.

G. *Discussion on VM allocation to Appropriate VM Process Results*

The test has shown that each case of the IFTS cloudlet/VM mapping process has provided a better result than mapping cloudlets to the available VM or mapping them randomly. We can summarize the findings of this test as follows:

- IFTS cloudlet to VM mapping process managed to decrease the failure rate from 28.08% in random mapping and 15.46% in available VM mapping to 5.91%, which is a huge difference and improvement.
- The results show that cloudlet and VM classification will improve the performance of the cloud by reducing the failure rate by 61.93% in random mapping and by 30.84% in available VM mapping.
- IFTS mapping process has increased the makespan by 1.14% in random mapping and 6.93% in available VM mapping, which is considered a cost of implementing FT in the cloud.

H. *Validation of IFTS Smart Decision on Application-Level Process*

The Algorithm 3 shows the steps of the smart decision process in IFTS at the application level. The process starts once the cloudlets are assigned to the VMs and ready to be executed. IFTS smart decision is a hybrid FT technique since it uses checkpointing and replication based on the priority of the cloudlet. The test compares the makespan and failure rate of IFTS with the following FT techniques:

- Replication: Cloudlets are replicated on any available VM. The replica will be delivered to the user in case the cloudlet fails.
- Checkpointing: periodic checkpoint is created, and a rollback procedure is performed when a fault is generated.

The simulation for this test requires fault injection at the application level. At a random time, the cloudlet status is set to fail. Once the status is changed, the Center for Internet Security (CIS) will notify the FT manager to check the selected FT policy assigned to the failed cloudlet. If the cloudlet is replicated, then the cloudlet replica will substitute the faulty cloudlet. In case the cloudlet has a saved checkpoint, the rollback to the last successful checkpoint is performed. The cloud configuration for this test is shown in Table XIX.

TABLE XIX. CLOUD CONFIGURATION FOR TESTING IFTS SMART DECISION PROCESS

Configuration	Unit
Number of Hosts	8
Number of Data Centers	1
Host Cores	8
MIP for Each Core	3000
RAM	2048 MB
Storage	100000 MB
Bandwidth	2000MB
Priority	Random value from 0.1 to 0.9
VM Count	20 VMs
Number of Cloudlets	100–500, different priorities

I. Results and Analysis of IFTS Smart Decision on Application-Level Process

Table XX shows the comparison results of implementing the IFTS smart decision on the application level compared with replication and checkpointing. Each value is an average result of a four-times simulation of each case.

TABLE XX. MEASURED RESULTS OF IFTS SMART DECISION PROCESS

#	Makespan in MS			Failure Rate %		
Cloudlets	Replication	Checkpointing	IFTS	Replication	Checkpointing	IFTS
100	1500	1350	990	3	2	1
200	1650	1435	1003	6	3	2
300	1800	1689	1230	10	6	4
400	1899	1745	1452	12	8	6
500	2009	1890	1745	14	11	7
Average	1771.600	1621.800	1284	9	6	4

The summary of our analysis of the above results is as follows:

1. The results for application-level FT show that IFTS has an improvement in the makespan and failure rate when compared with replication and checkpointing. Makespan is improved by 27.52% when using replication and by 20.83% when using checkpointing.
2. IFTS has managed to deliver more cloudlets successfully with an overall failure rate improvement of 55.56% compared with replication and 33.33% compared with checkpointing.

J. Discussion on IFTS Smart Decision on Application-Level Process Results

Based on the above results and analysis, we summarize our findings in the following points:

1. Using a smart decision based on priority will affect the performance of the cloud and will manage to decrease the failure rate.
2. When using replication only, the makespan is lower than checkpointing because, in case of failure, there is a replica that is being executed on another VM and it will be delivered to the user successfully; however, if the creation of the replica fails due to the lack of resources then the failure rate here will increase.
3. In checkpointing, the increase in the makespan is because of the rollback recovery process. The cloud will try to execute the cloudlet up to the last saved checkpoint. The main drawback of checkpointing is overhead and the required resources to store the checkpoints. As the cloudlet number increases, the number of checkpoints also increases and without controlling the checkpoint process, the failure rate will rise.
4. The improvement in IFTS comes from the fact that it balances the two FT techniques. However, it is highly dependent on the priority of the cloudlet.
5. IFTS uses the reserved resources to create the replica and checkpoints, which is considered an advantage that decreases the failure rate.

V. CONCLUSION AND FUTURE WORK

Nowadays, many organizations and businesses rely on cloud computing systems as using the cloud services comes with huge benefits, such as speeding up the delivery of IT systems and projects and lowering the operational cost of on-premises data centers. As a result of the increased demand for using cloud services, failures in delivering user requests also increase. Therefore, having a proper strategy to tolerate fault is crucial to comply with the SLA between the CSP and the cloud users. The cloud infrastructure has multiple levels and layers, each subject to failure. In this investigation, we have studied the cloud infrastructure and components and how the cloud processes user requests. After that, we proposed an Intelligent Fault-Tolerant Strategy (IFTS) that suggests an enhanced solution in each process of processing the user's request. Each IFTS step has been empirically tested and analyzed using the CloudSim simulator and compared with other relevant processes. All tests have shown that the proposed strategy has improved the reliability of the cloud. Here are some potential extensions of the work, including recommendations for future research directions and their corresponding applications for cloud computing systems:

- The proposed FT strategy is to be extended to include more application-level techniques rather than using only replication and checkpoint.
- The effect of the resource reservation for FT should be studied and the right percentage based on the current demand and available resources should be calculated.
- Currently, IFTS uses hybrid FT techniques and smart decisions based on priority; in future work, a different parameter can be used for the smart selection decision.
- In all experiments conducted to validate the proposed strategy, we have used the CloudSim simulator. However, in the future, it would be intriguing to validate IFTS by implementing it on real public cloud platforms.
- Lastly, this work will be expanded to include new enhancements to the strategy to enhance fault tolerance and cloud reliability.

APPENDIX A

Algorithm 1: Algorithm for VM Creation and Finding the Suitable Host

```
    Varibales: VMMapTable Map
    Input: HOSTs List , acceptedFailureRate double , VMType integer
    Output: Boolean Result (True, False)
1   Results = False;
2   IF (Virtual Machine not Created)
3       Sort (HOSTs List resources ascending);
4       For All (host available in HOSTs.List)
5           IF (host.type != VMTYpe or host.FailureRate >= acceptedFailureRate)
6               Continue For;
7           End IF
8           IF( host.freePE >= VM.neededPEs && host.FreeMemory
            >= VM.neededMemory && host.FreeIO >= VM.neededIO )
9               Index = index of host;
10          End IF
11          SelectedHost = HOSTS(Index);
12          Results = create VM on SelectedHost
12          IF(Results == True)
14              Update:
15                  VMMapTable( VM , selectedHost);
16                  Increase selectedHost.UsedPES(VM.neededPEs);
17                  Decrease SelectedHost.FreePES (VM.neededPEs);
18              Result = True;
19              Break;
20          End IF
21      End For
22  End IF
23  Return Results
```

APPENDIX B

Algorithm 2: Algorithm for Mapping Tasks to Appropriate VM

```
    Variables: ResearvedHost List
    Input: Cloudlets List , CloudletType integer , VMs List , acceptedFailureRate double
    Output: Boolean Result (True, False)
1   Results = False;
2   Do:
3       Sort (VMs List resources ascending);
4       For All (VM available in VMs.List)
5           IF (VM.type != CloudletType or VM.FailureRate >= acceptedFailureRate)
6               Continue For;
7           End IF
8           IF( VM.freePE >= Cloudlet.neededPEs && VM.FreeMemory
            >= Cloudlet.neededMemory && VM.FreeIO >= VM.neededIO )
9               Index = index of VM;
10              Result = True;
11          Else
12              FindSuitableHost (ReservedHost , acceptedFailureRate, CloudletType );
13              Break;
14          End IF
15          IF (Result != True)
16              For All (VM available in VMs.List)
17                  IF( VM.freePE >= Cloudlet.neededPEs && VM.FreeMemory
                    >= Cloudlet.neededMemory && VM.FreeIO >= VM.neededIO )
18                      Index = index of VM;
19                      Result = True;
20                  End For
21              End IF
22      End For
23  While (CloudLets.list is not finished);
24  Return Results
```

APPENDIX C

Algorithm 3: Algorithm for IFTS Smart Decision

```
    Variables: ResearvedHost List
    Input: VMs List , , priorityThreshold double
    Output: Boolean Result (True, False)
1   Results = False;
2   Do:
4       For All (VM available in VMs.List)
5           For All (cloudlet in VM.cloudlets)
6               IF ( cloudlet.prority > priorityThreshold)
7                   Submit cloudlet' to reserved host
                    Result = True
8               Else
9                   Create a periodic cloudlet checkpoint
10                  Result = True;
11              End IF
            End For
12      End For
13  While (CloudLets.list is not finished);
14  Return Results
```

CONFLICT OF INTEREST

The authors declare no conflict of interest.

AUTHOR CONTRIBUTIONS

The proposed IFTS design was developed by Dr. Kalim Qureshi. The implementation and results measurements are done by Taiba Yousef. The coding and measurement data validity checked by Dr. Kalim Qureshi. The paper editing is also done by Kalim Qureshi. The authors reviewed and revised the final version of the paper; all authors had approved the final version.

ACKNOWLEDGMENT

The authors thank Kuwait University for providing laboratory facilities to conduct this research.

REFERENCES

[1] H. Pallathadka, G. S. Sajja, K. Phasinam, M. Ritonga, M. Naved, R. Bansal, and J. Quiñonez-Choquecota, "An investigation of various applications and related challenges in cloud computing," *Materials Today: Proceedings*, vol. 51, pp. 2245–2248, 2022.

[2] T. Alam, "Cloud computing and its role in the information technology," in *Proc. IAIC Transactions on Sustainable Digital Innovation (ITSDI),* 2020, vol. 1, no. 2, p. 103.

[3] P. Kumari and P. Kaur, "A survey of fault tolerance in cloud computing," *Journal of King Saud University–Computer and Information Sciences*, vol 33, no. 10, pp. 1159–1176, 2021.

[4] Y. Sharma, B. Javadi, S. Weisheng, and S. Daniel, "Reliability and energy efficiency in cloud computing systems: Survey and taxonomy," *Journal of Network and Computer Applications*, vol. 74, pp. 66–85, 2016.

[5] B. Yadranjiaghdam, K. Hotwani, and N. Tabrizi, "A risk evaluation framework for service level agreements," in *Proc. 2016 IEEE*

International Conference on Computer and Information Technology (CIT), Fiji, IEEE, 2016.

[6] P. Guo and Z. Xue, "Real-time fault-tolerant scheduling algorithm with rearrangement in cloud systems," in *Proc. IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC),* Chengdu, China, IEEE, 2018.

[7] S. M. Abdulhamid, and M. S. Latiff, "A checkpointed league championship algorithm-based cloud scheduling scheme with secure fault tolerance responsiveness," *Applied Soft Computing,* vol. 61, pp. 670–680, 2017.

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

[9] A. Souza, A. V. Papadopoulos, L. T. Bolivar, D. Gilbert, and J. Tordsson, "Hybrid adaptive checkpointing for virtual machine fault tolerance," in *Proc. IEEE International Conference on Cloud Engineering,* 2018.

[10] M. G. Rakesh, A. Baunthiyal, and A. K. Jain, "Preemptive fault tolerance in DDS based distributed system using application migration," *IJRASET,* vol. 8, pp. 963–970, 2020.

[11] M. Hasan and M. S. Goraya, "Flexible fault tolerance in cloud through replicated cooperative resource group," *Computer Communications,* vol. 145, pp. 176–192, 2018.

[12] G. Yao, Y. Ding, and K. Hao, "Using imbalance characteristic for fault-tolerant workflow scheduling in Cloud systems," *IEEE Transactions on Parallel and Distributed Systems,* vol. 28, no. 12, pp. 3671–3683, 2017.

[13] E. AbdElfattah, M. Elkawkagy, and A. El-Sisi, "A reactive fault tolerance approach for cloud computing," in *Proc. 13th International Computer Engineering Conference (ICENCO)*, 2017.

[14] B. Ray, A. Saha, S. Khatua, and S. Roy, "Proactive fault-tolerance technique to enhance reliability of cloud service in cloud federation environment," *IEEE Transactions on Cloud Computing,* vol. 10, no. 2, pp. 957–971, 2020.

[15] J. Wang, W. Bao, and X. Zhu, "FESTAL: Fault-tolerant elastic scheduling algorithm for real-time tasks in virtualized clouds," *IEEE Transactions On Computers,* vol. 64, no. 9, pp. 2545–2558, 2014.

[16] M. Azaiez, W. Chainbi, and K. Ghedira, "Hybrid fault tolerance model for cloud dependability," in *Proc. 2019 IEEE 21st International*, 2019.

[17] M. Hasan and M. S. Goraya, "Fault tolerance in cloud computing environment: A systematic survey," *Computers in Industry,* vol. 99, pp. 156–172, 2018.

[18] T. Y. Alkhurafi, "An investigation of Intelligent Fault-Tolerant Strategy (IFTS) for cloud computing system", Ph.D. dissertation, Information Science Dept., Kuwait University, Kuwait, 2023.