CHAPTER ONE
INTRODUCTION

1.1 Background to the study

Computer advent has been a huge plus to human civil life as it provides both business and private world with an easy and fast means to generate and exchange information. Start-ups, medium and large-scale enterprises with local and international partners as well as transaction information are stored on respective remote computers. However, with the proliferation of networked devices and internet services, the amount of data and the frequency at which these data are produced have raised a major concern as the enormous amount of data generated has cost organisations a lot of money in the acquisition of highly functional local servers or cluster of computers capable of processing and housing the huge data. Nowadays, businesses and organisations are migrating to the cloud through cloud computing (CC) in order to obtain a high level of productivity at a considerable cost.

CC plays a significant role in providing computing services such as storage facilities, scalability, and more to businesses of varying sizes. It involves the collection of large pool of systems connected in private, public or hybrid networks, to provide dynamically scalable infrastructure for application, data and file storage (Nakkeeran, 2015). In cloud computing, the clients outsource their data to the cloud that performs the computing operations and store the results (Bawa, 2012). This differs from traditional computing where users use portable storage media and rely on machine robustness and speed. Rather, cloud computing offers a distributed environment that hosts hundreds of thousands of servers consisting of multiple memory modules, network cards, storage disks, and more. CC deliveries are categorized into three: Infrastructure as a Service (IaaS), Software as a Service (SaaS), and Platform as a Service (PaaS). IaaS offers virtualized environment where multiple users access computing architecture and resources such as server, data storage, and networking. PaaS provides an environment

where users can run, compile and program without the underlying infrastructural concern. While SaaS provides users with on demand pay per use software or applications. These service deliveries are cost effective, scalable and flexible. Despite the numerous benefits of CC, systems in providing these services, it is usually faced with reliability issues, and sometimes suffers from outages of services as a result of failures within cloud infrastructure. An outage is the case in which user's request is not completed in its desired deadline (Amoon, 2016). There are several types of failures that affect the reliability and availability of cloud services. Alshayeji *et al*. (2018), classified these failures into two groups: the request and execution failures. Request failure instances are overflow and timeout processes, which affect a request prior to the required resource access, while the execution failure affects a request during its execution time. Examples of execution failure include: data resource misses, computing resource missing, software, database, hardware and network failures. Both request and execution failures are peculiar in public clouds as enormous amount of services is provided due to the increasing demand thereby resulting to degradation of performance or availability.

However, failures can be tackled through the provision of fault tolerance in CC. Fault tolerance simply finds failures or faults in a system, and ensures with fault occurrence, the system still functions properly. Fault tolerant methods are majorly categorized into proactive and reactive fault tolerance based on failure handling techniques. The idea of proactive fault tolerance policies is to avoid the recovery from faults, errors and failures by predicting and proactively replacing them with other functional components (Kalanirnika and Sivagami, 2015). Some of the techniques that are based on proactive fault tolerance policies are Proactive Fault Tolerance (PFT) using preemptive migration, software rejuvenation, load balancing, and so on (Kalanirnika and Sivagami, 2015). Reactive fault tolerance on the other hand, reduces the effect of failures on application execution when the failure effectively occurs. Among the several types of reactive fault tolerance policies are checkpointing and replication techniques.

Checkpointing is a well-known fault tolerance technique that efficiently copes with soft errors. Unlike traditional time redundancy techniques, in which upon error detection, the program is restarted from the beginning, checkpointing stores checkpoints (intermediate states of the execution of the program), and when errors are detected, it forces the program to roll-back to the latest stored checkpoint. The advantage of this technique over other fault tolerance techniques is that it does not require a substantial amount of hardware redundancy, and it is effective for time-consuming and big applications. In CC, data replication allows several virtual machines (VMs) to have duplicates or copies of a request in order to avoid system starting from scratch when failure occurs (Amoon, 2016).

In general, computer systems are classified into real-time and non-real-time systems, depending on the requirement satisfying constraints. For real-time systems, the correct operation is defined as producing the correct output while satisfying a given time constraint (deadline). Depending on the consequences when deadlines are violated, real-time systems are divided into two namely, soft and hard real-time systems. In soft real-time system (SRS), the consequences are not very severe. A good example of a soft real-time system can be a mobile phone where eventual deadline violation results in a dropped call. On the other hand, violating the deadlines in the hard real-time system (HRS) usually result in catastrophic consequences. An example of an HRS system is the brake system control in a vehicle. They are also affected by soft errors, and therefore there is a need to employ fault tolerance as well. Although, special consideration should be taken when employing fault tolerance in real-time systems due to the fact that fault tolerance usually introduces time overhead.

However, applying single fault tolerant technique alone in the cloud have proven insufficient in providing reliability as researchers combine different fault tolerance techniques to ensure dependability of services in the cloud. In this research, checkpointing and replication are

applied in CC for both long and short tasks respectively for effective fault tolerance of the system.

Deng *et al*. (2012), proposed a fault tolerance and reliable computation in CC. The motivation of the research sprang from the need to provide reliable cloud computing system in the cause of fault. The research objective proposed a technique to improve the fault tolerance issue of cloud computing in scientific computation, while the research methodology involved matrix mathematical connotations. The research conducted by Deng *et al.* (2012), has the following limitations

a. The fault tolerance technique applied in the work was not stated.
b. The model was neither evaluated nor compared with existing methods.

Paul and Visuswasm (2012), worked on the research titled "Checkpoint-based intelligent fault tolerance for cloud service providers". The research was motivated by the need to achieve reliability for real-time computing on Cloud Infrastructure. The research objective proposed a smart checkpoint infrastructure for virtualized service provider and fault tolerance model for real-time computing. In the methodology, checkpoints are stored in Hadoop distributed files system as it allows fast resumption of task execution after a node crash. The method also improves fault tolerance since the checkpoints were distributed and replicated in all the nodes of the service provider. However, the research failed to consider any form of checkpoint overheads. That is, the cost, time, and frequency of replicating checkpoints in all the nodes of the service provider.

Kalanirnka and Sivaganmi (2015), presented a research on fault Tolerance in cloud using proactive and reactive techniques. The research was motivated by the need to provide a more reliable cloud computing system. The researchers proposed a reactive fault tolerance technique that uses checkpointing to tolerate faults. The methodology involved using virtual machine

(VM) framework to tolerate transient errors. The VM-μCheckpoint mechanism was implemented using CoW-PC (Copy on Write – Presave in cache) algorithm. The CoW-PC algorithm presaves all the tasks running on the VMs in a cache memory, on the occurrence of transient failure in VMs, it is noted and recovered using last presaved checkpoint from the cache memory. Once the tasks are executed successfully, the presaved checkpoints are deleted automatically from the cache. The research contributed to knowledge through the establishment of checkpointing technique to mitigate transient fault. However, the limitation of the work includes:

a. inability to include a proactive measure as proposed
b. the proposed model was not implemented in a real-time platform.

Guler and Ozkasap (2017), presented a "Compressed incremental checkpointing for efficient replicated key-value store". The research was motivated by the need to address the communication cost of the well-known primary-backup replication protocol. The objective of the research proposed compressed periodic incremental checkpoint algorithms to achieve improved throughput. The methodology involved setting up a replicated key-value store on geographically distributed nodes of the PlanetLab platform, and developed compressed incremental checkpointing algorithms to support primary-backup replication. The research contributed to knowledge by addressing the communication cost of primary-backup replication approach, and proposed periodic incremental checkpoint algorithms to improve performance. However, in their result, there was no significant increase in throughput as compared with the uncompressed checkpointing algorithm.

Sutaria *et al*. (2017), presented fault prediction and mitigation in CC. The research motivation was to effectively manage faults with the research objective focusing on a survey that illustrates defect prediction, mitigation method, technique and tools used for fault tolerance. The

methodology involved comprehensive review of literature. However, no new or modified fault tolerance technique was developed

1.2 Motivation

With high demand for cloud services and resources by users who migrated to the cloud in order to obtain high level of productivity, CC systems have experienced increase in outages or failures particularly in real-time cloud computing (RTCC). On the contrary, researchers have applied different fault tolerant techniques to mitigate failure. Some of these researches are not without their limitations.

Zaidi and Rampratap (2016), in their research titled "modelling for fault tolerance in cloud computing environment," presented a survey that deal with the understanding of fault tolerance techniques in cloud environment and on that proposed local and global checkpointing algorithm. The methodology involve d the detection of faults and its prevention by issuing checkpoints using round-robin task scheduler. Test cases were also designed for the validation of the proposed model. The research contributed to knowledge through the issuance of checkpoints using round-robin task scheduler. However, some shortcomings were observed. These are:

a. round robin is a queuing technique which average waiting time could be high during scheduling.
b. the model was not evaluated.

Devi and Paulraj (2017), presented a research work on "A multilevel tolerance in cloud environment." The motivation of the research was to reduce the impact of fault caused by increased demand of services in the cloud. They proposed a multilevel fault tolerance system in real-time cloud environment using reliability assessment algorithm and replication mechanism. The proposed model involved a two-level fault tolerance system. The first level was to avoid VMs failure at the logical layer hosted on the underlying physical component

through the use of reliability assessment algorithm. The second level was to ensure the high availability of task or data using replication mechanisms such as low latency fault tolerance (LLFT) messaging protocol and the leader determined membership protocol (LDMP). The proposed system contributed to knowledge by establishing a mechanism that tolerates faults in an efficient way by proving high reliability and availability. However, the proposed system was not evaluated to ascertain its performance.

Zhou *et al*. (2017), worked on enhancing reliability via checkpointing in cloud computing systems. The motivation of the research sprang from the failure of edge switches which may result to inaccessible of checkpoint images. The objective of the research proposed an optimal checkpointing method with edge switch failure aware. In the methodology, two algorithms were employed. The first algorithm employed the data centre topology and communication characteristic for checkpoint image storage server selection. The second algorithm employed the checkpoint image storage characteristic as well as the data centre topology to select the recovery server. The simulation results show that the proposed checkpoint-based method can efficiently ensure service reliability and reduce the root layer network resource consumption in cloud computing systems. However, proposed technique suffers checkpoint latency due to fixed interval between nodes.

Benoit *et al*. (2019) also worked on combining checkpointing and replication for reliable execution of linear workflows with fail-stop and silent errors. The motivation of this work was to resolve large-scale platforms with recurrent experience of errors from two different sources: fail-stop errors and silent errors. It was expected that the solution will minimize the execution time of linear workflows in error-prone environments. Benoit *et al* adopted an optimal dynamic programming algorithm of quadratic complexity and simulation to solve both problems. It was able to establish gain over the checkpoint-only approach especially when checkpointing is

costly and error rates are high. However, the work only considers task duplication and triplication and the model was not implemented in a real-time cloud computing environment.

In summary, the shortcomings of the reviewed literature, among others, are:

a. high average waiting time in round robin in Zaidi and Rampratap (2016)
b. inability of Devi and Paulraj (2017) to evaluate their model
c. checkpoint latency in Zhou *et al*., (2017) as a result of fixed intervals
d. limited task replication (maximum of three) in Benoit *et al*. (2019)

Both checkpointing and replication techniques have been comprehensively studied separately in the works of Visuwasam, (2012), Zhou *et al.* (2017), and Chinnathambi *et al*. (2019). Also, some researchers have combined both in few cases, as seen in Sun, *et al.,* (2012). However, no researcher has endeavoured to combine the two in the context of real-time cloud service delivery. Therefore, this research is motivated by the need to resolve the aforementioned limitations through hybridization of checkpointing and replication techniques to improve reliability and availability of RTCC systems.

1.3 Objectives

The specific objectives of the research are to:

a. design a failure-mitigating hybrid model for cloud environment using checkpointing and replication techniques;
b. implement the model designed in (a); and
c. evaluate the performance of the developed model based on standard performance metrics.

1.4 Methodology

The proposed system comprises four (4) layers namely the user, task controller, fault detector and fault tolerance layers. In the user layer, user U submits a request R along with quality of service (QoS) of parameters S via cloud clients to the controller layer as follows:

$$U = \{R, S\}: R = \{T_1, T_2, T_3, .., T_n\} \text{ and } S = (t, c, c_r) \quad 1.1$$

where R is a request that involves set of tasks T (such as inventory taking, data analysis, language processing), n represents the total number of tasks in R, and S represents QoS parameters with t as turnaround time, c as monetary cost, and c_r as computing resources. The controller houses the service manager (S_m) and the fault tolerance scheduler module (F_m). The module ensures that user's request is achievable as it checks the database (D_b) that holds information such as computing capacity (c_c), storage capacity (s_c), usage history (u_h) and failure history (f_h) as represented in Equation 1.2.

$$S_m = f(D_b) \quad 1.2$$
$$\text{where } D = \{c_c, s_c, u_h, f_h\}$$

The Virtual Machine (VM) should be capable of handling such request, otherwise the request is discarded. Thereafter, as presented in Equation 3, the request is passed to the scheduler module which contains sub-modules namely virtual machine and fault tolerance scheduler (F_m).

$$S_{sm} = \{VM_i, F_m\} \qquad for\ i = (1, 2, 3, \ldots, n) \quad 1.3$$

The VM scheduler gets the list of stable VMs to handle the user's request by accessing the database that holds VMs information, afterwards, it performs reliability checks on the VMs based on their probability failure rate as presented in Equations 1.4 and 1.5.

Given a list of VM_i, the probability of an instance VM_i in a given execution time is expressed as:

$$P_i(f) = \frac{e^{-\mu}\mu^f}{f!} \qquad 0 \leq P_i(f) \leq 1 \quad 1.4$$

$$\mu = \frac{f_i}{T_i/\tau_{ji}} \quad 1.5$$

where $P_i(f)$ represents the failure probability of VM_i, $f = (f_1, f_2, f_3, \ldots f_n)$ is the number of failures in a given time, μ is the average number of failures in VM_i, and f_i is the number of

failure of VM_i. T_i represents the time taken for f_i failures to occur, and τ_{ji} represents the estimated time for a task

j to be executed on VM_i. However, the probability for a failure to occur (i.e. i = 1 in $P_i(f)$) during the execution of a task is expressed (from Equation 1.4) in Equation 1.6.

$$P_i(f_{i=1}) = \mu e^{-\mu} \quad 1.6$$

After obtaining the failure probability of all *VM*s, the machines will be ranked according to failure probability values in ascending order. Fault tolerance scheduler module's responsibility is to determine the appropriate fault tolerance module to be used via an FTScheduler algorithm that checks the list of virtual machines available and makes appropriate selection based on the algorithm presented as follows:

FTScheduler Algorithm

```
Begin
  T := {ti} ∀ i ∈ Z
   for ti ∈ T
      VMi := {Li} ∀ i ∈ Z
        if VM > 1
          VM := R # R is Replication
        else
          VM := C # C is Checkpointing
        end if
    end for
End
```

The fault tolerance layer contains two fault tolerance techniques namely: replication and checkpointing. The replication module offers fault tolerance by making replicas of the request on the most appropriate VM in the list which is the VM with the least $P_i(f)$. However, the number of replicas (ρ) for the VMs is determined dynamically as stated in Equation 1.7:

$$\rho = \begin{cases} \rho_i = 1, \ if \ P_i(f) \leq k \\ \rho_i = 2, \ \ otherwise \end{cases} \quad 1.7$$

where the value of k is determined based on the availability of VMs in the list, and R is the number of replicas for a request. The checkpointing module tolerates faults by saving request data to allow resumption from that point later, in the case of failure. Here, the request execution

is partitioned in segments for optimal checkpoints n_c^* which is expressed as a function estimated error probability Q_{est} based on non-equidistant checkpointing as adapted in Equation 1.8.

$$n_c^*(Q_{est}, T_{adj}) = -(ln(1 - Q_{est})) + \sqrt{(ln(1 - Q_{est}))^{T_{adj}} - \frac{2 * T_{adj} * (ln(1 - Q_{est}))}{\tau}} \quad 1.8$$

Here, T is the processing time and τ represents checkpoint overhead. Q_{est} is estimated periodically by employing a history unit that keeps track of the number of successful executions N_s and the number of erroneous probabilities N_e as defined in Equation 1.9.

$$Q_{est} = \frac{N_e}{N_e + N_s} \quad 1.9$$

The proposed model will be simulated using CloudSim, and evaluated based on throughput, response time, robustness and trend.

1.5 Organisation of thesis

The rest of this thesis is organized as follows: Chapter Two presents the related works and extensively reviewed existing literature, to investigate existing loopholes and justify the need to carry out this research. Chapter Three discusses the methodology used in the design and the overall analysis of the system. Chapter Four presents the implementation and results and evaluation while Chapter Five concludes the research with recommendations drawn from this research and the contributions made to knowledge.