CHAPTER THREE

SYSTEM DESIGN

3.1 Preamble

Performing benchmarking experiments in repeatable, dependable, and scalable environments using real-world cloud environments is difficult because of lack of the necessary infrastructure that are needed. Hence, the need for simulation is important as a proof of the concept (Vecchiolaa, *et al.,* 2011). Simulation can be a powerful technique for evaluating the performance of large-scale cloud computing services in a relatively low costs low risk and time-sensitivity (Bendechache *et al*., 2019). Simulation of cloud environment is extremely challenging because cloud exhibit varying demands, supply pattern and resources. Several simulators in existence were investigated under distributive systems. These include; GangSim developed in (Dumitrescu and Foster, 2005), SimGrid by (Casanova, *et al.*, 2008) or OptorSim developed in (Bell *et al.*, 2002).While these are good tools however, literature reveals that they cannot work in a real time cloud environment. This is because the cloud simulator requires good learning algorithms and should provide good application programming interfaces like java to build and train the network in cloud based applications (Qasem and Madhu, 2018). Hence, the use of CloudSim was adopted in this research.

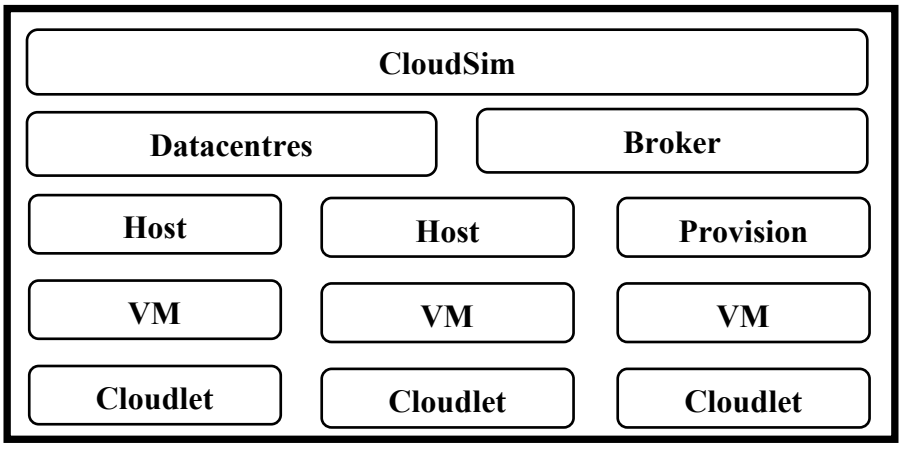

Figure 3.1: Basic architecture of CloudSim

3.2 Functionalities of the simulator

A datacentre can manage several hosts which in turn manages VMs during their life cycles. The datacentre must have some characteristics which must be basically for the host so each datacentre may have some host, for instance, if one host in the datacentre and this particular host will have some kind of hardware configuration. Host is a CloudSim component that represents a physical computing server in a Cloud: it is assigned a pre-configured processing capability (expressed in millions of instructions per second - MIPS), memory, storage, and a provisioning policy for allocating processing cores to VMs.

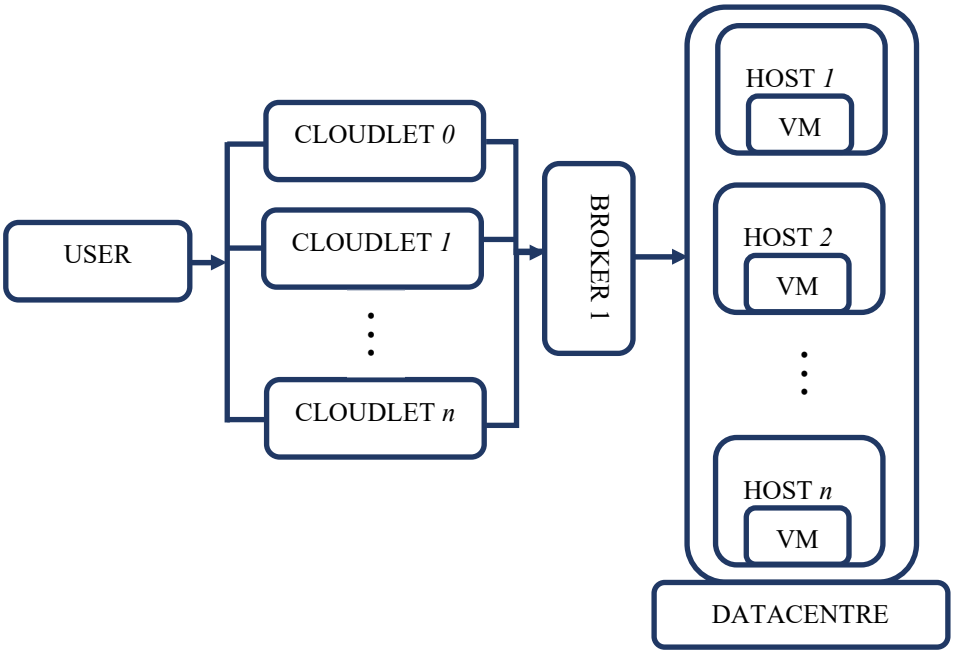

Figure 3.2: CloudSim data flow

3.2.1 Fundamentals of task execution with and without checkpointing

Checkpointing is a commonly used technique for reducing the execution time of long running programs in the presence of failures. With checkpointing, the status of the task under execution is saved intermittently. Upon the occurrence of a failure, the task execution is restarted from the most recent checkpoint rather than from the beginning. Due to its overhead, checkpointing may not always be beneficial, and, if it is, an optimal checkpointing strategy may be determined so as to

minimize the expected execution time or to maximize the probability of the execution time not exceeding a critical limit.

i. Task execution without checkpointing

The execution of a task with a given processing resource as measured by the computation time on a failure-free system in the presence of failures. It is assumed that a given task is equipped with a fault-tolerance mechanism that enables it to restart after the occurrence of failures. Without checkpointing, the program has to be restarted from the beginning each time a failure occurs. Normally, the execution of a task will complete when its entire processing resources are completed without failures. The time of execution of a task is defined as the time elapsed from start to finish. This includes periods of repairs and reprocessing after failures.

If x is the processing resource of the task and $T(x)$ be its execution time of the task. (Note that the task processing resource is identical to its execution time under failure-free environment.) Assuming failures occur according to a Poisson process at rate γ. (Poisson Distribution is usually used especially when failures are caused by many different but independent sources. Also, as seen in some works, it is assumed that failures are detected immediately they occur. When a failure occurs, there is a repair time, which is denoted by a random variable R. Ordinarily, it is assumed that no failures may occur during the repair time R. Clearly, due to the randomness of the failure/repair process, $T(x)$ is a random variable, even though the processing resource x is fixed.

For a random variable, say R, the probability distribution function (CDF) is denoted by

$$F_R(t) = P(R) \leq t, \tag{3.0}$$

and its probability density function (pdf) by $f_R(t)$, and its Laplace-Stieltjes transform (LST) by

$$\emptyset_R(s) = E(e^{-sR}) = \int_{t=0}^{\infty} e^{-st} f_R(t)dt \tag{3.1}$$

Also, the CDP for $T(x)$, is given as $F_T(t,x)$ its pdf is $f_T(t,x)$ while its LST is given as $\varphi_T(s,x)$. This gives a closed form expression for the LST of the program execution time, $T(x)$, in the presence of failures.

Without checkpointing, the LST of the task execution time in the presence of failures is given by:

$$\emptyset_T(s,x) = \frac{(s+\gamma)e^{-(s+\gamma)x}}{s+\gamma(1-\emptyset_R(s))(1-e^{-(s+\gamma)x})} \quad 3.2$$

If H is taken to be the time to the first failure after starting the execution of the task, the conditioning on H = h gives

$$T(x)|_{H=h} = \begin{cases} x, & if\ h \geq x \\ h + R + T(x), & otherwise \end{cases} \quad 3.3$$

If $h \geq x$, then the task will complete in x units of time. If $h < x$, then a failure occurs before the completion of the program. In this case, there is a repair time R after which the task execution is restarted from the beginning, again requiring x units of uninterrupted processing time to complete. The LST of $T(x)$ therefore gives rise to:

$$T(x)|_{H=h} = E\left(e^{-sT(x)}\middle|H=h\right) = \begin{cases} e^{-sx}, & if\ h \geq x \\ e^{-sh}\emptyset_R(s)\emptyset_T(s,x) & otherwise \end{cases} \quad 3.4$$

then, unconditioning on H, this becomes:

$$\emptyset_T(s,x) = \int_{h=0}^{\infty} \emptyset_T(s,x)\,|_{H=h}\gamma e^{-\gamma h}dh$$

$$= e^{-(s+\gamma)x} + \frac{\gamma\emptyset_T(s,\ x)(1-e^{-(s+\gamma)x})}{s+\gamma} \quad 3.5$$

as previously stated.

The LST in Equation 3.2 can be inverted numerically or by inspection to obtain the pdf $f_T(t,x)$. In most cases though, numerical inversion is necessary and can be carried out using one of many procedures for the numerical inversion of Laplace transforms. The first and higher moments of

$T(x)$ is obtained from its LST but the interest of this aspect is the expected execution time of the task, $E(T(\ x))$, (without checkpointing) in the presence of failures is expressed below:

$$E\big(T(x)\big) = \left(\frac{1}{\gamma} + E(R)\right)(e^{\gamma x} - 1) \qquad 3.6$$

This shows that $E\big(T(x)\big)$ grows exponentially with the processing resource *x*.

ii. Task execution with checkpointing

Here, checkpointing is considered to avoid excessive execution times due to restarts in the presence of failures. The effect of checkpointing is examined on the distribution of task execution time and its expectation. Given the processing time T of the task, that is, the time needed for a task to complete when no error occurs during the execution and considering that n_c checkpoints are to be used, then, the length of a single execution segment t_{ES} is given as:

$$t_{ES} = \frac{T}{n_c} \qquad 3.7$$

From equation 3.7, when the number of checkpoints is high, the overhead due to re-execution of erroneous execution segments is reduced because the execution segments are shorter. On the other hand, when the number of checkpoints is low, the overhead due to checkpointing decreases because the checkpointing operations are performed less frequently.

3.2.2 Equidistant checkpointing

The optimality of checkpointing strategies with fixed (equal) productive time between checkpoints has been established at the system and the task levels. It is assumed that the task is divided into equal parts and a checkpoint is placed at the end of each task part for instance, then, the analysis of the task execution time is considered and failure during a fixed checkpoint duration are taken into account.

3.2.3. The execution time with equidistant checkpoints

Here, the LST of the execution time of a task is derived when checkpoints are equally spaced with respect to the processing time. An expression for its expectation with respect to the number of checkpoints in the program is also obtained. If x is the total processing resource of the task, which is divided into n equal parts and represented as $T(x,n)$ being the total execution time. It is important to note that there is checkpoint at the end of each task except the last segment. This implies a total of n - 1 checkpoints. The following gives an expression for the LST of $T(x,n)$

With $n-1$ equidistant checkpoints, the LST of the task execution time in the presence of failures is given by:

$$\emptyset_T(s,x,n) = \left(\frac{(s+\gamma)\emptyset_c(s+\gamma)e^{-(s+\gamma)\frac{x}{n}}}{s+\gamma\left(1-\emptyset_R(s)\left(1-\emptyset_c(s+\gamma)e^{-(s+\gamma)\frac{x}{n}}\right)\right)}\right)^{n-1}\left(\frac{(s+\gamma)e^{-(s+\gamma)\frac{x}{n}}}{s+\gamma\left(1-\emptyset_R(s)\left(1-e^{-(s+\gamma)\frac{x}{n}}\right)\right)}\right) \qquad 3.8$$

This can be exemplified as follows:

Each of the first n - 1 task parts requires $\frac{x}{n}+C$ units of uninterrupted processing time to complete. The execution times of the first n - 1 task parts are independent and distributed random variables with each given as $T\left(\frac{x}{n}+C\right)$ where C, the checkpoint duration, is a random variable; however, it is fixed for a given task part. The LST of $T\left(\frac{x}{n}+C\right)$ is given as follows

$$\emptyset_T\left(s,\frac{x}{n}+C\right) = \frac{(s+\gamma)\emptyset_C(s+\gamma)e^{-(s+\gamma)\frac{x}{n}}}{s+\gamma\left(1-\emptyset_R(s)\left(1-\emptyset_C(s+\gamma)e^{-(s+\gamma)\frac{x}{n}}\right)\right)} \qquad 3.9$$

The last task segment only requires $\frac{x}{n}$ units of uninterrupted processing time that is, without a checkpoint and its execution time will be $T\left(\frac{x}{n}\right)$. The Laplace-Stieltjes transform (LST) of $T\left(\frac{x}{n}\right)$ is obtained as follows:

From equation 3.2, that is, $\emptyset_T(s,x) = \frac{(s+\gamma)e^{-(s+\gamma)x}}{s+\gamma(1-\emptyset_R(s))\left(1-e^{-(s+\gamma)x}\right)}$

$$\emptyset_T\left(s,\frac{x}{n}\right) = \frac{(s+\gamma)e^{-(s+\gamma)\frac{x}{n}}}{s+\gamma\left(1-\emptyset_R(s)\left(1-e^{-(s+\gamma)\frac{x}{n}}\right)\right)} \quad 3.10$$

The total task execution time is the sum of n independent random variables corresponding to the execution times of the n parts. The first n - 1 of the identically distributed and having the LST of $\emptyset_T\left(s,\frac{x}{n}+C\right)$, and the last one having the LST of $\emptyset_T\left(s,\frac{x}{n}\right)$

This implies that:

$$\emptyset_T(s,x,n) = \left(\emptyset_T\left(s,\frac{x}{n}+C\right)\right)^{n-1}\left(\emptyset_T\left(s,\frac{x}{n}\right)\right)$$

$$\equiv \left(\frac{(s+\gamma)\emptyset_C(s+\gamma)e^{-(s+\gamma)\frac{x}{n}}}{s+\gamma\left(1-\emptyset_R(s)\left(1-\emptyset_C(s+\gamma)e^{-(s+\gamma)\frac{x}{n}}\right)\right)}\right)^{n-1}\left(\frac{(s+\gamma)e^{-(s+\gamma)\frac{x}{n}}}{s+\gamma\left(1-\emptyset_R(s)\left(1-e^{-(s+\gamma)\frac{x}{n}}\right)\right)}\right) \quad 3.11$$

As usual, in equation 3.8 or 3.9, the LST can be inverted numerically to obtain the Probability density function $f_R(t,x,n)$. Again, the interest is the expected task execution time $E\big(T(x,n)\big)$, that is, with n-1 equally spaced checkpoints, the expected task time of execution in the presence of failures is

$$E\big(T(x,n)\big) = \left(\frac{1}{\gamma}+E(R)\right)\left((n-1)\left(\emptyset_C(-\gamma)e^{\gamma\frac{x}{n}}-1\right)+\left(e^{\gamma\frac{x}{n}}-1\right)\right) \quad 3.12$$

3.2.4 Optimal number of checkpoints

The overall goal of checkpointing is to reduce the average execution time (AET) by reducing the time overhead. Time overhead is in two parts: one part is the overhead caused by performing the checkpointing operations while the other is the overhead caused by re-executing the failed segments. Invariably, these two parts contribute and are related to the number of checkpoints. This means that when the number of checkpoints is large, the overhead increases because the checkpointing operations are performed more frequently and the overhead due to re-execution of

erroneous execution segments is reduced because the execution segments are shorter as depicted in equation 3.7, that is, $t_{ES} = \frac{T}{n_c}$. On the other hand, when the number of checkpoints is small, the overhead due to checkpointing decreases because the checkpointing operations are performed less frequently and the overhead due to re-execution of erroneous execution segments increases because the execution segments are longer.

3.2.5 Poisson distribution

Poisson distribution is used, among other usages, to evaluate the probability of an isolated event occurring a specific number of times in a given time interval, for instance, number of faults or failures at a time interval. With respect to fault tolerance of the virtual machines, events occur at random and also applies to the useful life of the system components, the VMs. The general expression for Poisson distribution is given in equation 3.13:

$$P_x(t) = \frac{(\lambda t)^x e^{-\lambda t}}{x!} \qquad 3.13$$

where $P_x(t)$ is the probability of event occurring *x* times in time *t* and,

λ is the failure rate

Expected value of Poisson distribution is given as,

$$E(x) = \mu = \lambda t \qquad 3.13$$

and the probability of zero failures in time *t* (that is when *x* = 0), is given as:

$$P_0(t) = \frac{(\lambda t)^0 e^{-\lambda t}}{0!} = e^{-\lambda t} \qquad 3.14$$

3.3 System architecture

The developed system is composed of 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 QoS of parameters S represented in the relation 1.0 via cloud clients to the controller layer.

$$U = \{R, \mathrm{S}\}: R = \{T_1, T_2, T_3, .., T_n\} \text{ and } S = (t, c, c_p) \quad 1.0$$

where R is a request that involves set of tasks T, 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_p as computing power.

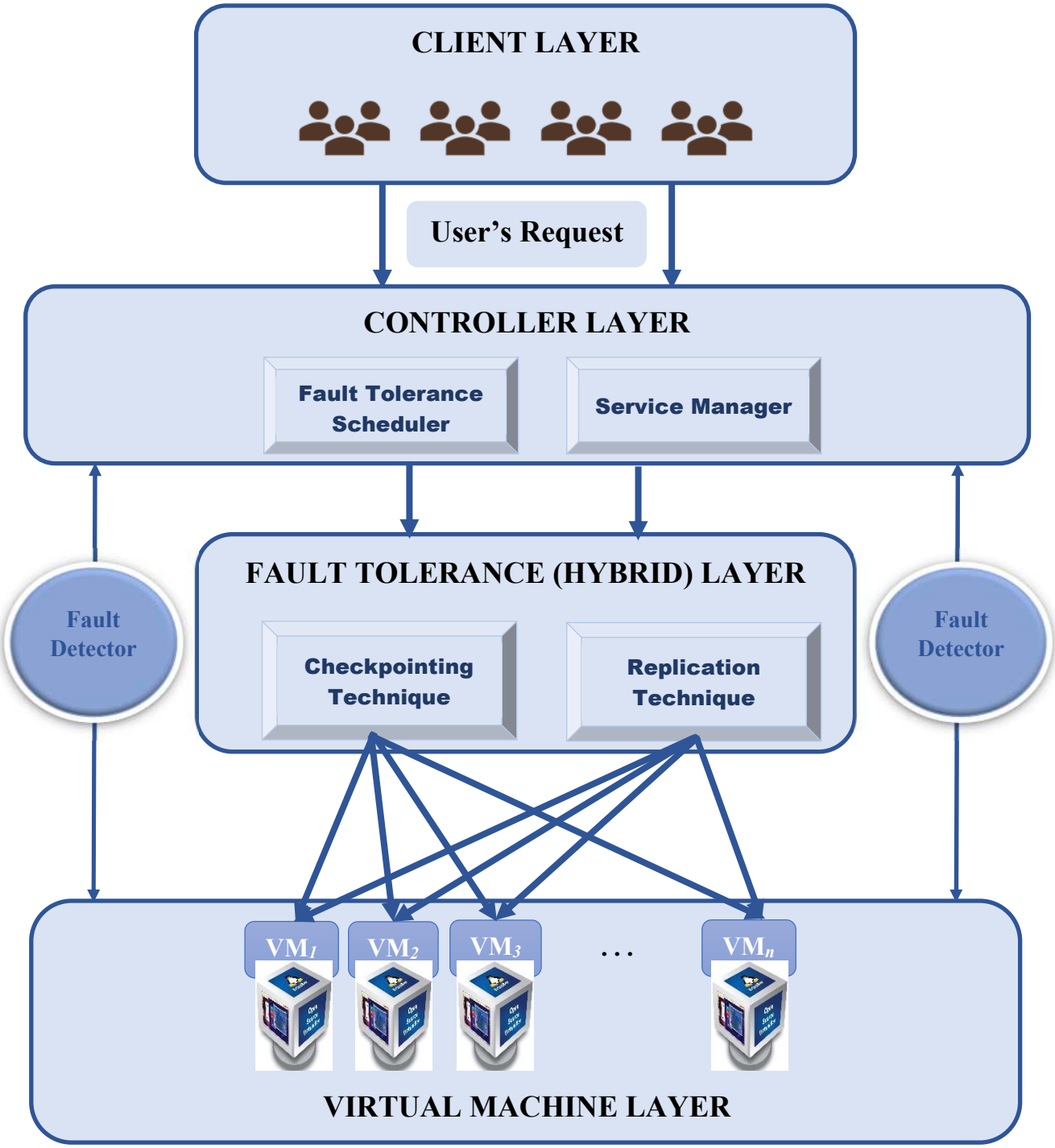

Figure 3.3: System Architecture

The user layer is the user layer that permits users interaction with the cloud. The user layer consists of cloud clients. These are computer hardware or software that rely on computing for application delivery as well as delivery of cloud services. Therefore, user's requests are sent from the user layer via cloud clients.

The controller houses the service manager (S_m) and the scheduler module. The S_m ensures that user's request is achievable as the module checks the database that holds information such as computing and storage capacities in addition to usage and failure history, as represented in equation 3.15.

$$S_m = f(D_b) \tag{3.15}$$

where $D_b = \{c_c, s_c, u_h, f_h\}$

c_c is the computing capacity; s_c is the storage capacity, u_h is the usage history and f_h failure history

The *VM*s should be capable of handling such request, otherwise the request is discarded. Thereafter, the request is passed to the scheduler module which contains sub-modules (S_{sm})namely virtual machine scheduler (VM_s) and fault tolerance scheduler (FT_s).

$$S_{sm} = \{VM_s, FT_s\} \qquad \forall\ s \in S$$

The *VM* scheduler gets the list of stable *VM*s to handle the user's request by accessing the database that holds VMs information, it then performs reliability checks on the *VM*s based on their probability failure rate as in equations 3.16 and 3.17.

Given a list of $VM_i, i = (1,2,3, \ldots., n)$, the probability of an instance VM_iin a given execution time is expressed as:

$$P_i(f) = \frac{e^{-\mu}\mu^f}{f!} \qquad 0 \leq P_i(f) \leq 1 \tag{3.16}$$

$$\mu = \frac{f_i}{T_i/\tau_{ji}} \tag{3.17}$$

where $P_i(f)$ represents the failure probability of VM_i, $f = (f_1, f_2, f_3, \dots, f_n)$ 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 during the execution of a task is expressed in equation 3.18 as:

$$P_i(f_{i=1}) = \mu e^{-\mu} \qquad 3.18$$

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 algorithm that checks the list of VMs available and makes appropriate selection based on certain criteria as follows:

appropriate selection based on certain criteria as follows:

```
Begin
  T:= {t_i} ∀i∈Z
   for t_i ϵ T
        VM_i := {L_i} ∀i∈Z
                  if VM> 1
                    VM := R # R is Replication
                  else
                    VM := C # C is Checkpoint
              end if
    end for
End
```

The S_m is responsible for ensuring the achievement of QoS requirements. In this research, the considered QoS requirements include time and monetary costs. A user can submit his request to the cloud through this module along with the QoS requirements. The module will search for the availability of appropriate *VMs* that can carry out the user request. If the search indicates the presence of appropriate *VMs*, the Service Inspector will accept the request and it will deliver it to the fault tolerance Scheduling module. Otherwise, the request will be discarded.

The fault tolerance layer contains two fault tolerance modules namely: replication and checkpointing modules. 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 for the VMs is determined dynamically as stated in equation 3.19:

$$Replication = \begin{cases} R_i = 1, if\ P_i(f) \leq k \\ R_i = 2,\ if\ P_i(f) > k \end{cases} \quad 3.19$$

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 tolerate faults by saving request data to allow later resumption from that point in the case of failure. Here, the request execution is partitioned in segments and provides optimal check points n_c^* which is expressed as a function estimated error probability Q_{est} using non-equidistant checkpointing in equation 3.20

$$n_c^*(Q_{est}, T_{adj}) = -(ln(1 - Q_{est})) + \sqrt{(ln(1 - Q_{est}))^{T_{adj}} - \frac{2 * T_{adj} * (ln(1 - Q_{est}))}{\tau}} \quad 3.20$$

where T is the processing time, 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 3.21.

$$Q_{est} = \frac{N_e}{N_e + N_s} \quad 3.21$$

In order to find the best mathematical proof of concept to support VM scheduling and real time cloud service request processing, various literatures were searched. The researcher adopts that of Barthelemy and Janowitz, (1991) and George, (1954) based on similar assumptions. The assumptions are that:

i. Cloud service providers seek to service requests as soon as it is made, without violating SLAs
ii. Service request arrival rate is random

iii. Service is random that is, as long as a request is being serviced by a functioning VM, the request must be completed during an interval of j. j is considered the exponential holding time before the VM Scheduler finds a stable VM.

The probability that a VM scheduler finds a faulty VM during an interval j is given in equation 3.22 as:

$$P_n(t) = aj + o(j) \tag{3.22}$$

where a is the average arrival rate.

Assuming the average processing rate of all VMs in the datacentres are equal, and if they were to service customers' request independently it would be L.

Let $P_n(t)$ be the probability that at time t, there are n customer requests in the system,

Let r be the probability that only stable VMs can service a request at a particular time h .

Then:

$$P_n(T+h) = P_{n-1}(T)ah + P_n(T)(1-2rLh)(1-ah) + P_{n+1}(T)(2rLh) + o(h) \tag{3.23}$$

Thus, $$\frac{dP_n(T)}{dT} = aP_{n-1}(T) - (2rL + a)P_n(T) + 2rLP_{n+1}(T) \tag{3.24}$$

If $n = 0$,

$$P_o(T+h) = P_0(T)(1-ah) + P_1(T)(2rLh) + o(h) \tag{3.25}$$

$$\frac{dP_0(T)}{dT} = -aP_0(T) + 2rLP_1(T) \tag{3.26}$$

Statistical equilibrium is assumed so that

$dP_n(T)/dT = 0, (n = 0,1,2, \dots)$ and also,

Assumes that $\lim_{T\to\infty} P_k(T) = p_k$ has a unique value for all $k = 0,1,2, \ldots$

Then, $$p_1 = \frac{a}{2rL} p_0, \quad 3.27$$

$$p_2 = (\frac{a}{2rL})^2 p_0, \quad 3.28$$

$$p_k = (\frac{a}{2rL})^k p_0 \quad 3.29$$

From $\sum_{i=0}^{\infty} p_i = 1$,

$$p_0[1 + \frac{a}{2rL} + \left(\frac{a}{2rL}\right)^2 + \cdots \Big] = 1 \text{ ; provided } a < 2rL \quad 3.30$$

Then $$p_k = \frac{a^k(2rL-a)}{(2rL)^{k+1}} \quad 3.31$$

$k = (0, 1, 2, \ldots)$ will give the probability that the VMs are in a particular state (Stable or Faulty) upon a request arrival.

Using the above mathematical models, the probability that at a particular time (*h*), there are *n* numbers of requests waiting for the VM Scheduler could be calculated.

3.4 Checkpointing and replication

In order to sustain generality, the failure model of this research is based on assumptions in (Ling *et al.*, 2001), (Chen *et al.*, 2010) and (Sun *et al.*, 2012) as depicted in figure 3.4. The assumptions are as follows:

i. Failures in this model are short-lived failures, instead of permanent failures. So, the failures can be recovered by checkpointing. Failures will occur randomly during the running of the system, but a failure can be immediately detected when it occurs
ii. Failure will not happen during the system recovery period.

iii. The checkpointing interval δT is a constant, which means that failure time T_{fail} is always less than the checkpointing interval δT. Meanwhile, the checkpointing over- head time chkT is also a constant in a real time cloud system.

iv. The failed system always can be recovered from the last full checkpointing, which means the failure recovery time T_{rec} is always less than the checkpointing interval δT.

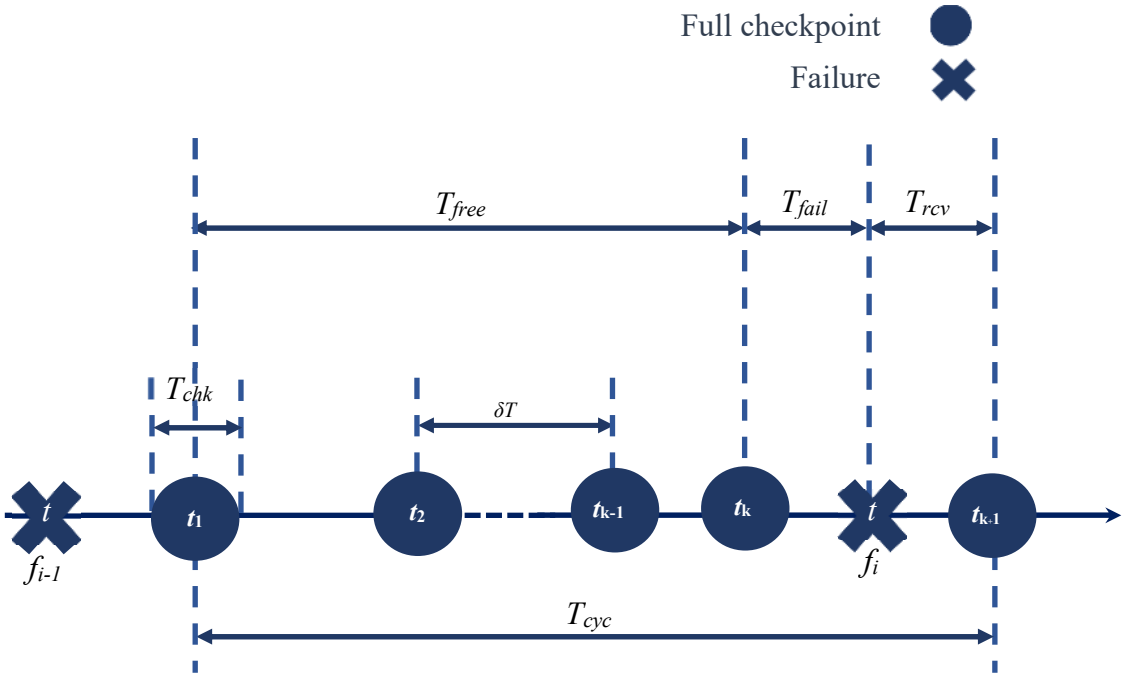

Figure 3.4: Failure cycle of Checkpointing Technique Source: (Sun *et al*., 2012)

Table 3.1: List of notations

Notation	Description
s	The time of the k-*th* checkpointing
t_{fi}	The time of the i-*th* failure occurrence
δT	The time interval between the consecutive checkpoints
ρ(t)	The checkpointing density function of a continuous checkpoint
T_{cyc}	The time interval of a failure cycle
T_{fail}	The time interval between the failure point T_f and the last checkpointing tk
T_{rcv}	The time interval between the failure point t_f and the new checkpointing f_{k+1} after system recovery
T_{chk}	The time interval; used to set a checkpoint
T_{free}	The checkpointing overhead during the longest failure free time interval of the consecutive checkpoints
$f(t)$	The failure density function
F(t)	The failure distribution function
E(t)	The expectation function of the number of failures

The failure probability of a Virtual Machine is assumed to follow Poisson Distribution. This implies that the number of failures in any two different periods of time is independent over the time changes. The failure probability distribution of a virtual machine i in a given time interval can be expressed as:

$$F_i(X) = \frac{e^{-\mu}\mu^x}{x!} \quad 0 \leq F_i \leq 1 \; and \; x = 0, 1, 2, \dots n \tag{3.32}$$

Where X $(x_0, x_1, x_2 \dots, x_n)$ represents the number of failures that can happen in a given time period and μ, is the average number of failures in the given time period for a virtual machine i. The value of μ is given in equation 3.33 by:

$$\mu = \frac{f_i}{T_i / t_{ji}} \tag{3.33}$$

where f_i represents the number of failures of a virtual machine i, and T_i is the period of time in which f_i failures have occurred. t_{ji} represents the estimated time when service request j is

executed on VM i. The probability of one failure ($x = 1$) to take place during the execution of a request is therefore given by:

$$F_i(x_1) = \mu e^{-\mu} \quad 3.34$$

3.5 Checkpointing module and algorithm

The checkpoint module is responsible for providing checkpoints at intervals while considering checkpoint overheads such as checkpoint size and frequency. That is, the time expended by resources while generating checkpoints. If the frequency of generating checkpoints is high, it can amount to excessive time and resources consumption. Hence, the checkpoint module will regulate the frequency and size of checkpoints by assigning tasks with high QoS requirements with more checkpoint frequencies. The checkpoint frequency is characterized by the sequence of checkpointing intervals.

$Z_x(j), j \geq 1$ determines the instant of time at which checkpointing occurs before a cloud service request can be completed. The failure rate of the system is dependent on the value of these intervals. In the new model, checkpoints are taken during processing time D_i. It is assumed that failure in service delivery is independent of the present state of the VMs, therefore failure F_{i+1} can occur.

i. before the first checkpoint, and

ii. after the first checkpoint.

These two (2) conditions were adopted from (Tantawi & Ruschitzka, 1984), and taken into consideration in designing the new model as shown in Figure 3.5 and 3.36 respectively.

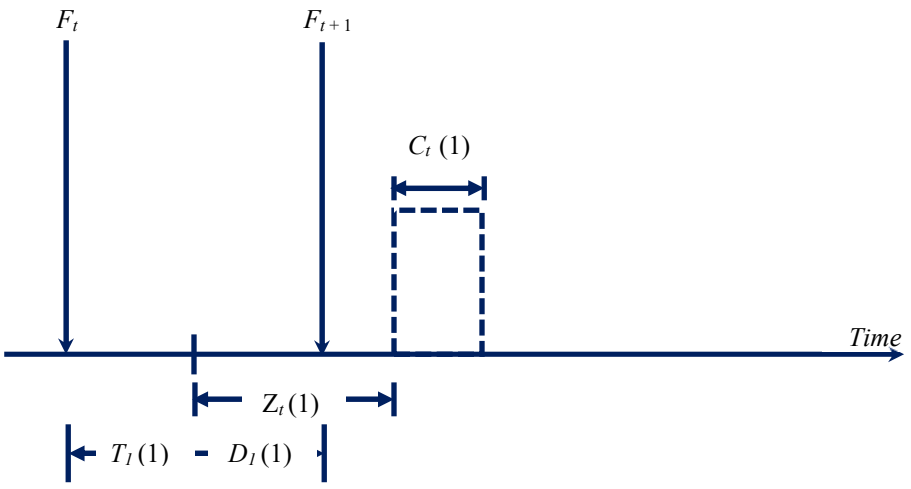

Figure 3.5: First case (failure before the first checkpoint)

In the first case (before the first checkpoint), the checkpointing interval $Z_i(1)$ is larger than the normal processing time D_i as shown in Figure 3.5. The reprocessing time Y_{i-1} associated with failure F_{i+1}, is thus the sum of the uncheckpointed production time D_i and the reprocessing time Y_{i-1} associated with F_i.

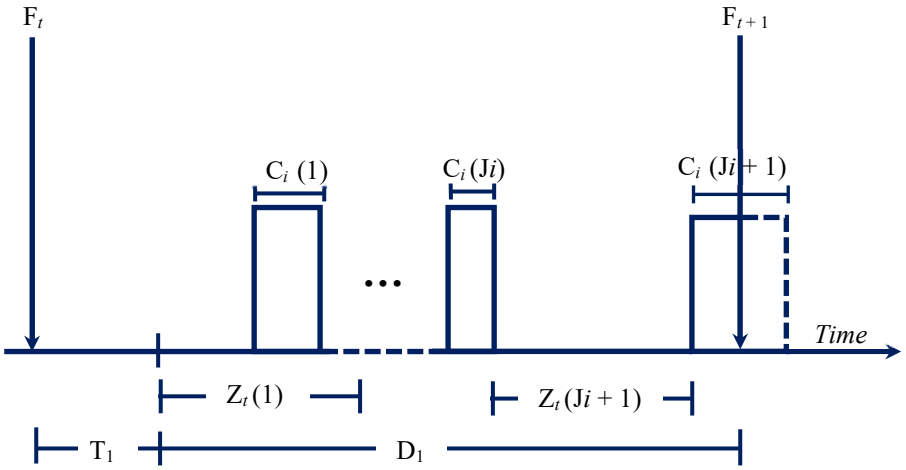

Figure 3.6: Second Assumption (After Checkpointing)

The third case is illustrated in Figure 3.6, where the reprocessing time Y_i associated with failure F_{i+1} is the interval between the completion of the J_ith checkpoint and the time of failure.

Let $V_i(j)$ be defined to be the cumulative service time between the end of the total failure recovery time and the jth checkpoint, during the *i*th cycle,

$$V_i(j) = \begin{cases} \sum_{k=1}^{j} Z_i(k), & j= 1,2,3,... \\ 0, & otherwise \end{cases} \quad 3.35$$

Let $B_i(j)$ denote the sum of the first chekpointing times during the *i*th cycle

$$B_i(j) = \begin{cases} \sum_{k=1}^{j} C_i(k), & j= 1,2,3,... \\ 0, & otherwise \end{cases} \quad 3.36$$

It follows that the starting time $S_i(j)$ and the ending time $E_i(j)$ of the jth checkpoint during the jth checkpoint during the ith cycle equals:

$$S_i(j) = V_i(j) + B_i(j-1), \quad j = 1, 2 ..., \quad 3.37$$

and

$$E_i(j) = V_i(j) + B_i(j-1), \quad j = 1, 2 ..., \quad 3.38$$

Respectively.

Note that time in the above equations is measured from the end of the total failure recovery time.

3.5.1 Checkpointing algorithm

The execution time of task j on VMi

The remaining execution time of task j on VMi

$F_i(x_1)$: Failure probability of VMi

$F_i(x_0)$: Probability of no failure on VMi

h: Checkpoint interval

z: Number of failures during task execution

Calculate $F_i(x_1) - \mu e^{-\mu}$;

For each task j allocated to VMi, do

{

$z = 0$;

$h = T_{ji} X F_i(x_z)$;// Initial checkpoint interval

$Tr_{ji} = T_{ji}$;

Start execution of j on i;

Do

{

$Tr_{ji} = Tr_{ji} – h$

If failure occurred, then

{

Z++;

$Tr_{ji} = Tr_{ji} + h$;

$h = h(1 – F_i(x_z))$; // decrease checkpoint interval

Restore last checkpoint;

Restart execution from Tr_{ji} - Tr_{ji};

}

At time T_{ji}- Tr_{ji} perform a checkpoint;

$h = h(1 – F_i(x_z))$; // increase checkpoint interval

Resume execution;

3.6 Replication and replication module

The replication module offers fault tolerance by making duplicates of the application on all virtual machine instances. In this research, an approach by Amoon (2016) is employed to choose valuable VMs (that is, VMs that will greatly affect the cloud if failure occurs) as duplicating on all VMs will result in high consumption of resources. The approach uses the concept of failure probability to decide VMs that are most valuable.

Given a set of VMs $VM = \{1, 2, 3, \dots, n\}$, the probability failure of an instance VM_i in a given time is expressed as:

The replication model consists of two (2) phases. The first phase determines which data files should be replicated on the VM while the second phase determines how many suitable VMs should hold replicated data based on the following algorithm:

3.6.1 Replication algorithm

$F_i(X)$: $The\ failure\ probability\ of\ a\ virtual\ machine\ i$

P_i: The percentage of cloud profit gained through usage of virtual machine i

Rep: The number of replicas

$F_i(X)(k)$, k = 0,1,2, ... ,n, are integers such that 0<= $F_i(X)(k)$<=1.0 and $F_i(X)(0)$<$F_i(X)(1)$< ... .$F_i(X)(n)$

$P_i(y)$, y = 0,1,2, ... ,m, are the percentage of cloud profit gained by the Virtual machine i , such that 0<= $P_i(y)$<=100 and

$P_i(0)$< $P_i(1)$< ... . $P_i(m)$

Rep(l) (w), l = 0,1,2, ...n and w= 0,1,2, ..., m are integers

***For** (a = 0 ; a < n; a++)*

{

***For** (b = 0; b < m; b++)*

{

If (*$F_i(X)(a)$<=$F_i(X)$< $F_i(X)(a+1)$ and $P_i(b)$ <= P_i< $P_i(b + 1)$)*

Rep = Rep(a)(b);

}

}