Reliable and Energy-efficient Resource Provisioning and VM Consolidation in Cloud Computing Environment

Yogesh Sharmaa,b,∗, Bahman Javadia, Weisheng Sia, Daniel Sunb

aSchool of Computing, Engineering and Mathematics
Western Sydney University, Australia
bSoftware and Computational Systems, DATA61 — CSIRO, Australia

Abstract
VM consolidation is an important technique used in cloud computing systems to improve energy efficiency. It migrates the running VMs from under utilized physical resources to other resources in order to reduce the energy consumption. But in a cloud computing environment with failure prone resources, focusing solely on energy efficiency has adverse effects. If the reliability factor of resources is ignored then the running VMs may get consolidated to unreliable physical resources. This will cause more failures and recreations of VMs, thus increasing the energy consumption. To solve this problem, this paper proposes a failure-aware VM consolidation mechanism, which takes the occurrence of failures and the hazard rate of physical resources into consideration before performing VM consolidation. We proposed a failure prediction technique based on exponential smoothing to trigger two fault tolerance mechanisms (VM migration and VM checkpointing). A simulation based evaluation of the proposed VM consolidation mechanism was conducted by using real failure traces. The results demonstrate that by using the combination of checkpointing and VM migration with the proposed failure-aware VM consolidation mechanism, the energy consumption of cloud computing system is reduced by 34% and reliability is improved by 12% while decreasing the occurrence of failures by 14%.

Keywords: Cloud Computing, Failures, Reliability, Energy Consumption,

∗Corresponding author
Email addresses: y.sharma@westernsydney.edu.au (Yogesh Sharma), b.javadi@westernsydney.edu.au (Bahman Javadi), w.si@westernsydney.edu.au (Weisheng Si), daniel.sun@data61.csiro.au (Daniel Sun)
1. Introduction

In the ongoing era of information and communication technology, cloud computing has emerged as a break through computing paradigm which uses Virtualization technology to provide a powerful and flexible computing environment. Unlike traditional computing paradigm, cloud computing provides a ready to use platform to run the applications without worrying about the management of underlying infrastructure. This brings the businesses a good opportunity to minimize their operational expenses by bringing down their infrastructure administration costs. Due to these reasons businesses are switching to cloud computing at a great pace, which projects the future of the technology very productive and bright. Among 1002 organizations ranging from SMBs to large enterprises considered in a report published by RightScale in 2017 [1], 80% of them are already using cloud services and 14% are planning to adopt it. According to the Forbes, the worldwide cloud revenue will increase from $98 billion in 2016 to $167 billion by 2020\(^1\) and if this trend will continue then it will further rise to 186 billion by 2026. The projected revenue growth of cloud computing services (SaaS, PaaS and IaaS) has been shown in figure 1. Users of cloud computing access the service using easy-to-use portals such as AWS management console without being known about the underlying system. But while providing such an abstract view of the system, cloud computing systems have to perform many complex operations besides managing a large underlying infrastructure. Due to such complex operations, cloud computing systems confront cloud providers with many challenges such as security, sustainability, reliability, management etc. [2]. Among all the challenges, reliability and energy consumption are the two key challenges that needs careful attention and investigation.

In this study, reliability is considered as the probability with which an application/task will finish the execution before the occurrence of a failure. A failure in the services of a cloud costs significantly to both providers and customers. The report [3] from Ponemon institute in 2016 revealed that

\(^1\)http://www.forbes.com/sites/louiscolumbus/2016/03/13/roundup-of-cloud-computing-forecasts-and-market-estimates-2016/#5128807074b0
the average down-time cost of data centers due to outages is approximately $740,357 per year with an average of $9000 per minute. According to the Information week, each year IT outages result in the revenue loss of more than $26.5 billion [4]. Due to increasing the acceptance of cloud services, data center size is also increasing at a large pace which further increases the failure cost such that failure cost increases in proportion to data center size (figure 2). The energy requirement to operate the cloud infrastructure is also increasing in proportion to the operational costs. Approximately, 45% of the total operational expenses of IBM data centers goes in electricity bills. It has been estimated that the servers mounted in Microsoft’s cloud based data-centers consumes around 2 terawatts-hours (TWh) of energy per year for which the company pays approximately $2.5 billion per year as electricity bills [5]. Apart from the operational costs, huge amount of energy consumption by cloud computing infrastructure causes huge amount of carbon and green house gases emission in the environment.

This is evident that to provide the failure free or reliable environment to the users to run their applications, cloud computing service providers accommodate the redundant/back-up resources. If the primary resource will fail then the back-up resource will begin to run the user’s application to provide them a failure free environment. Along with the redundancy, cloud comput-
ing providers also deploys the buffer resources to accommodate the workload spikes (Microsoft is adding 10,000 servers to its data centers every month) but most of the time these servers sit idle or underutilized, which increases the operational expenses of cloud computing infrastructure by increasing the energy consumption. According to McKinsey and Company, the average utilization of resources/servers in cloud based data centers rarely exceeds than 6% and 30% of the servers (3.6 million in U.S and 10 million worldwide) are sitting idle and consuming electricity without performing any useful work [6]. At the system level, 50% of the energy consumed by the systems sitting in idle state. This unoptimized energy consumption by the cloud based data centers makes them a big consumer of electricity which has been projected to reach 140 billion kWh by 2020 which was 91 billion kWh is 2013 [7]. The key technique used to reduce the energy consumption is by running the resources on a low power scaling level or by turning off the under-utilized or idle resources such as back-up by migrating the running virtual machines from the under-utilized resources to other resources.

To maximize the reliability of the cloud computing services, all the cloud service vendors add back-up resources/hosts (hosts, nodes and resources are being used interchangeably in this work) and use replication as well as load balancing as fault tolerance mechanism. Adding extra resources increases
energy consumption more steeply than the reliability. The key technique used to reduce the energy consumption is by running the resources on a low power scaling level or by turning off the under-utilized or idle resources such as back-up by migrating the running virtual machines (VM) from the under-utilized resources to other resources. Turning off the back-up resources will reduce the reliability of the system. For example, in the case of VM consolidation (key technique to reduce energy consumption in cloud computing systems), if a physical machine fails due to some hardware or software issues before the completion of tasks and there are no recovery resources, then all the VMs and their corresponding processes will have to start again. This will dramatically increase overheads such as energy consumption and resource utilization. This creates a critical trade-off between these two metrics (figure 2). In this work, a framework of cloud computing system has been provided which uses Reliability and Energy aware resource provisioning and VM allocation policies [8]. Both reactive (checkpointing) and proactive (VM migration) mechanisms have been used to provide fault tolerance. To reduce the energy consumption dynamically, Reliability aware VM consolidation mechanism has been proposed. To further reduce the overheads of all the adopted fault tolerance and energy optimization mechanisms, failure prediction has also been incorporated. Following are the contributions of this

![Figure 3: Reliability and Energy Efficiency Tradeoff](image)
work

1. Mathematical models have been provided to calculate the reliability, energy consumption and finishing time of the tasks incorporating overheads incurred because of the occurrence of failures and VM migrations while executing the tasks in a failure prone cloud computing environment.

2. Resource and VM management policies using the proposed models to optimize the reliability and energy-efficiency of cloud computing systems while tolerating the occurrence of failures using failure prediction and VM migration.

3. Failure-aware Energy efficiency VM consolidation mechanism to reduce the energy wastage while considering reliability factors of the physical machines.

The remainder of the paper is organized as follows: Section 2 gives a brief survey about the existing work in the related area. Section 3 explains the system architecture used in this work followed by the application, deadline, reliability and power models. Section 4 presents the description about failure prediction and fault tolerance mechanisms and corresponding mathematical formulations. Section 5-6 includes the formulation of finishing time and energy consumption while executing the tasks in the presence of failures. In section 7, resource and VM management policies using the proposed models are presented. Section 8 consists of the details about the system set-up, workload model, performance evaluation metrics and reports the results in graphical form.

2. Related Work

Reliability and energy efficiency of cloud computing systems are the important objectives that the research community has been focused on. This section covers the works that have been recently done in reliability, energy consumption and both together in cloud computing systems.

Ao Zhou et al. [9] has targeted the data transfer delay and network resource consumption problem while recovering from failures using K-fault tolerance replication mechanism. A three step solution consisting of recourse allocation, VM placement and VM recovery has been provided. For VM recovery, specific solutions have been provided for software and hardware failures by taking VM proximity into consideration in order to minimize the data
loss and reduce processing delays. With the same objective of maximizing the reliability and minimizing the execution delays while using replication for fault tolerance, Guoqi Xie et al. [10] have proposed four heuristic algorithms to optimize the computing cost along with the aforementioned metrics. Unlike [9], authors have used complex work-flow applications to evaluate the proposed algorithms. Though the proposed methods provide high fault tolerance using replication but the cost in terms of resource usage and energy consumption has been ignored. However, we have used other fault tolerance mechanisms such as checkpointing and VM migration because of high usage cost of replication while ensuring high reliability and minimizing energy consumption. We have also explored heterogeneity and dynamic nature of cloud computing systems by employing different hardware types and VM consolidation and also used real time failure traces to inject failures in the simulated cloud computing environment. In comply to the aforementioned work, further a good amount of work which either focused on reliability or energy efficiency issues in cloud computing can be found in the literature. But very limited work has been done for achieving a global maxima by combining them. Besides the work being covered in the following part of the section, a survey of the state of art in reliability and energy efficiency mechanisms has been provided in our previous work [7].

It has been claimed that as the operating frequency of a system increases wrt to supply voltage, reliability of the system increases but energy efficiency decreases [11]. This makes the task scheduling a challenge to achieve these two contradictive goals at the same time. In response to the challenge three different algorithms such as SHRHEFT, SHRCPOP and SHREERM have been proposed by Longxin Zhang et al. [12]. Dynamic voltage scaling and shared recovery technique have been used to regulate the energy consumption and to ensure the reliability, respectively. After performing the experiments, it has been concluded that SHREERM algorithm surpasses the rest. With the same objectives, a genetic algorithm (BOGA) for task-scheduling to optimized the reliability and energy-consumption of high performance computing systems have been proposed by the same authors [13]. The performance of the proposed algorithm has been compared with other two algorithms such as modified MODE and MOHEFT and the superiority of the new algorithm has been shown over the other algorithms. All the approaches proposed in [12] [13] are focused specifically on high performance computing systems. It has also been assumed that at most one failure will occur during the life time of a task. In our work, this assumption has been rejected with the injection
of multiple failures.

Xiwei Qiu et al. [14] provides a theoretical correlation model for the fine grained measurement of reliability, power consumption and performance of cloud computing systems. A frequency scaling based power model while considering maximum CPU utilization has been used where as we have formulated the power consumption based on variable utilization while operating CPU at maximum frequency. The proposed work has been analysed numerically except the reliability model which has been simulated. However, our evaluation of the proposed formulation has been done both analytically and by using simulation though only simulation based results have been reported.

While focusing on reliability, computing energy and cooling energy, X. Li et al. [15] have proposed two line-based scheduling algorithms i.e. Ella-W and Ella-B. Mathematical models to rank the physical machines in terms of reliability and energy consumption have also been provided and metrics combining all the factors have been proposed to evaluate the algorithms. Similar to our work, authors have done the simulation based study of the proposed methods using real failure and workload traces. In contrast to the proposed work, we are using checkpointing and VM migration with and without failure prediction to tolerate failures besides the failure and energy aware measures being taken during the resource provisioning and allocation phase. We have also used time series analysis to predict the occurrence of failures and also exploit the dynamic nature of cloud computing systems by employing reliability aware VM consolidation to regulate the energy consumption dynamically.

Amir Varasteh et al. [16] have studied the interplay of energy consumption and reliability while performing the VM consolidation in cloud computing systems. A fine grained mathematical model has been provided to minimize the total data center cost while regulating the energy consumption and reliability. The proposed model has been analysed using Matlab based simulation where as we have shown the interplay of both metrics using simulation based results using real life data. Authors have calculated the reliability of homogeneous systems as the function of system activity (on-off cycles) where as we have calculated the reliability under the occurrence of failures in a heterogeneous environment. Authors have used the random VM allocation to the physical machines to test their models. However, we have provisioned physical resources by considering both reliability and energy consumption attributes together.
3. System Architecture

The targeted cloud computing environment in this study has been shown in Figure 1. System consists of a pool $P$ of failure prone heterogeneous resources/nodes. From the resource pool, resources get provisioned to run the heterogeneous VMs executing the tasks arriving at a specific rate. The system has been divided into four different layers. The bottom layer consisting of the resources such as servers with different hardware configurations on top of which the VMs are running. All the VMs running on a server are managed by a VM manager installed on the server. The number of VMs running on a node will not exceed the number of cores available on the node. One core will be allocated to each VM and VMs are not allowed to share the cores with each other (such configuration can be obtained in Xen [17] hypervisor). Memory of the node is being shared by the running VMs and to avoid the interference during run time, each VM has an exclusive share of the host memory. Decision about the allocation and migration of VMs are taken by the Resource Management System (RMS). The RMS is the heart of the architecture where all the reliability and energy aware resource pro-
visioning, allocation and VM migration policies are incorporated. The role of RMS is to gather the parameters from the energy management and fault management modules and takes the decision about the VM allocation and migration so that the reliability of the system will be maximized and energy consumption will be minimized. Users/Brokers are submitting their tasks to the RMS seeking execution before the deadline. On the arrival of new tasks, the current status of the resources and resource requirements of new tasks get evaluated at RMS. On the basis of the evaluation, optimized decisions about the resource provisioning and VM allocation takes place according to the proposed algorithms to regulate the reliability and energy-efficiency of the system. Failure prediction module has also been managed by the RMS. On the basis of the prediction results, running VMs get migrated from a suspicious host to some other more reliable host.

3.1. Application Model

Cloud computing systems can be used to run a variety of applications or workloads consisting of dependent tasks such as Map-reduce applications or independent tasks such as Bag-of-Task applications. In this work, Bag-of-Task (BoT) application composed of bags or groups of independent and sequential compute intensive tasks has been used because of their wide adoption in scientific and commercial organizations such as Facebook [18]. The most common examples of BoT applications are image manipulation applications (astronomy, image rendering, video surveillance), data mining applications, Monte Carlo simulations and intensive search applications. Each BoT consists of set of independent tasks, \( T = \{t_i \mid 1 \leq i \leq n\} \). Every task \( t_i \) has a corresponding length, \( l_i \). In this work, the length of a task has been referred to the number of instructions. Each task \( t_i \) will be allocated to a VM \( vm_j \in VM \), where \( VM = \{vm_j \mid 1 \leq j \leq m\} \) and each VM, \( vm_j \) runs a set of tasks, \( \Gamma_j \), where \( \Gamma_j \subseteq T \). To launch the desired number of VMs, a number of physical machines or nodes, \( N = \{n_k \mid 1 \leq k \leq x\} \) gets provisioned from resource pool. Every task has a deadline \( d_i \) associated to it which has been calculated according to the model proposed in the following section.

3.2. Deadline Model

Due to the occurrence of failures, fault-tolerance mechanism overheads (Section IV) and dynamic nature of cloud computing systems, execution delay happens, which results in SLA violations and affects the quality of service
being provided. In order to tolerate the impact of all the aforementioned factors while fulfilling the conditions of SLA, deadline $d_i$ corresponding to each submitted task $t_i$ has been calculated using equation 1. Defining the deadline gives the provider a time window corresponding to each task within which the task can be finished while fulfilling the SLA conditions and without any service interruption being faced at the user end.

$$d_i = \begin{cases} s_i + (f.l_i), & \text{if } [s_i + (f.l_i)] < c_i \\ c_i, & \text{otherwise} \end{cases}$$ (1)

$s_i$, $l_i$ and $c_i$ are the starting time, task length and completion time of a task $t_i$, respectively [19]. $f$ is the stringency factor that defines the deadline strictness i.e. higher the value of $f$ is, higher deadline relaxation the task has. Rather than rejecting a task for a deadline miss (hard deadline), the soft deadline concept has been adopted which means it reduces the value of the computation for the users [20]. More the execution of a task will be delayed, more the value will be reduced. In case of missing a deadline, the remaining value $\vartheta_i$ of a task $t_i$ has been calculated as follows

$$\vartheta_i = \begin{cases} 1 - \left(\frac{c_i - d_i}{d_i}\right), & \text{if } c_i > d_i \\ 1, & \text{otherwise} \end{cases}$$ (2)

### 3.3. Reliability Model

The proposed reliability model is based on the utilization of the system. Figure 5 shows the failure count per hour for a host in cluster G1/S8/C1 of Grid5000 failure traces taken from Failure Trace Archive [21], an online repository of failure traces gathered from various sites. It has been observed that during the working hours from 7 am to 7 pm, the number of failures are higher than the non-working hours. It can be interpreted as a correlation between the occurrence of failures and system activity/utilization. During the working hours the utilization/activity of the systems is higher than the non-working hours which increases the occurrence of failures. The same conclusion has been drawn in [22] [23] [24]. On the basis the obtained correlation between the system utilization and occurrence of failures, it is concluded that the failure rate of a system/node/host depends on its utilization level which is a function of time, i.e., varying according to the time.

This work is mainly focused on the physical resource failures in order to bring the evaluation of reliability in accordance to other factor, such that,
energy consumption, which is the hard-wired parameter of physical resources. Terms node/host/physical resources/system are used interchangeably. In this work, a VM running on a physical resource/node only failed, when the node failed. The failure rate/hazard rate, $\lambda_{jk}$ of a $vm_j$ running on a node, $n_k$ with utilization $u_k$ is calculated as follows

$$\lambda_{jk} = \lambda_{max} \times u_k^\beta$$

where, $\lambda_{jk}$ is the failure rate of a VM $vm_j$ running on a node $n_k$ with utilization $u_k$ and $\lambda_{max}$ is the failure rate of node $n_k$ at maximum utilization. $\beta (> 0)$ is the sensitivity factor which shows the sensitivity of the failure rate towards the utilization. When $\beta = 1$, it shows the linear relationship of $\lambda_{jk}$ with $u_k$, which is a function of time, i.e., varying according to time [8] [23]. In this work, it is assumed that all the running VMs shares the maximum hazard rate similar to the physical node $n_k$ which they are running on. As failures in cloud computing systems are inevitable, every node $n_k$ in the resource pool has a Mean Time between Failures ($MTBF_{max}$) at maximum utilization $u_{max}$, which is calculated by the RMS empirically from the node failure history.
Hazard rate or Failure rate ($\lambda_{\text{max}_k}$) for a node $n_k$ at maximum utilization is calculated as

$$\lambda_{\text{max}_k} = \frac{1}{MTBF_{\text{max}_k}}$$  (4)

The failure rate is assumed to be following Poisson distribution [11][12] and remains constant for each utilization level. So, the probability with which a VM, $vm_j$ running on a node $n_k$ with utilization $u_k$ with hazard rate $\lambda_{jk}$ will be able to finish the execution of all the running tasks is equal to the probability with which the longest task, $l_{\text{max}_j}$ running on $vm_j$ will finish the execution before the occurrence of a failure.

$$R_{vm_{jk}} = \exp^{-\lambda_{jk} \times l_{\text{max}_j}}$$  (5)

This probability is called reliability of a VM. As stated earlier, all the VMs running on a node get failed when host fails. This shows that the node and corresponding VMs share serial relationship with each other. So the reliability of a node $n_k$ while running $m$ VMs is calculated as the product of the reliabilities of all the VMs as follows

$$R_k = \prod_{j=1}^{m} (R_{vm_{jk}})$$  (6)

However provisioned nodes fail independently [14]. Between the provisioned nodes, neither the serial relationship nor parallel relationship exists. So the reliability of the system is calculated as the average of the reliability values possessed by all the provisioned nodes at a particular instance.

### 3.4. Power Model

Every node in the resource pool has the power profile information such that the power consumption at minimum utilization, $p_{\text{min}}$ and power consumption at maximum utilization $p_{\text{max}}$. To calculate the power consumption by a VM $vm_j$ with utilization $u_j$ running at node $n_k$ following model [25] is used

$$P_k(u_j) = (\frac{k}{k} \times P_{\text{max}_k}) + ((1 - \frac{k}{k}) \times P_{\text{max}_k} \times u_j)$$  (7)

where, $\frac{k}{k}$ is the ratio of $P_{\text{max}_k}$ and $P_{\text{min}_k}$. The power model used to calculate the energy consumption of the system in this work considers the power consumed by CPU only because it has been argued that CPU is the biggest consumer of the power among other devices such as memory units and storage systems [26] [27].
4. Failure Prediction

Fault tolerance methods in cloud computing systems are divided into two
classes: reactive and proactive methods [7]. In reactive methods, the whole
effort is to recover the failed tasks as soon as possible with minimum over-
heads whereas in proactive methods, the emphasis is to avoid the occurrence
of failures. In Chapter ??, a reactive fault tolerance method, checkpointing is
used to recover the failed tasks. In this work besides checkpointing, a proac-
tive fault tolerance mechanism, VM migration is adopted to run the tasks
without failing by migrating the running VMs from a physical resource to
other healthy and more reliable resources. To identify the physical resources
expecting to be failed and to schedule the VM migrations, failure prediction
is used. In order to predict the failures, an average based time series analysis
method called Exponential Smoothing [28] is used. The reason for choosing
the average based prediction method is the inconsistency and stationarity of
the available data, which is collected from Failure Trace Archive [21]. On
the basis of the patterns found in the traces, parametric models such as
ARIMA models [29] were tried to fit to predict the occurrence of failures but
the Mean Square Error (MSE) was found to be higher than MSE of average
based method. In order to predict the occurrence of failures, the prediction of
Time between Failures (TBF) is targeted. To predict \( n \) failures, predictions
of \( n - 1 \) TBFs is done.

Among all the average based prediction methods such as simple aver-
age, weighted average and k-period moving average [29], the exponential
smoothing is chosen because it considers all the time series values to make a
prediction with associated weights. Though this is true for weighted average
as well, exponential smoothing has a formal method to calculate the weights
corresponding to each contributing value of the time series. Suppose there
is a set of \( n \) TBFs corresponding to a host \( n_k \), \( TBF_k = \{tbf_t \mid 1 \leq t \leq n \} \). So the forecast corresponding to a \( tbf_{t+1} \) by using exponential smoothing is
calculated as follows

\[
(tbf_k)''_{t+1} = \begin{cases} 
\alpha \times (tbf_k)'_t + ((1 - \alpha) \times (tbf_k)''_t), & \text{if } n > 1 \\
(tbf_k)'_t, & \text{otherwise}
\end{cases}
\]

where, \( (tbf_k)'_t \) is the actual value of TBF between two consecutive failures at
time \( t \) and \( (tbf_k)''_t \) is the forecasted value obtained at time \( t - 1 \) for time \( t \)
for a node $k$. $\alpha$ is the smoothing constant i.e. $0 < \alpha < 1$. Value of $(tbf_k)^n_1$ is taken as the simple average of the time series.

5. VM Migration based on Failure Prediction

As mentioned in section III, VM migration is adopted as a fault tolerance mechanism which gets triggered on the basis of the prediction results. VM migration can further be divided into two types: stop-and-copy VM migration and live VM migration [30]. Due to less down time and less migration overheads, live VM migration is adopted in this work.

5.1. VM Migration Overheads

Although VM migration has many advantages over other reactive and proactive fault tolerance mechanisms, small overheads in terms of execution time get imposed on all the tasks corresponding to a VM because of the interruptions while performing migrations. The overheads can vary according to the configuration of a VM such that memory usage and type of application that VM is executing. Among several live VM migration approaches [31], the pre-copy migration approach is adopted because of its less migration and downtime overheads. In pre-copy migration, the memory pages of a running VM, $vm_j$ gets copied to the destination host, iteratively. The approach works with an assumption that at some point the memory pages required to get copied will be small enough so that the $vm_j$ can be stopped and migrated to the destination host. This ensures the minimum memory page errors and less downtime. The proposed migration overhead model is based on the model provided by Sherif et al. [32]. The total migration overhead, $TMO_{ij}$ for a task $t_i$ running on $vm_j$ is the sum of the migration overheads, $MO_{ij}$ incurred during $n$ pre-copy iterations and downtime overheads, $DTO_{ij}$.

$$TMO_{ij} = MO_{ij} + DTO_{ij}$$

(9)

The migration overhead, $MO_{ij}$ is calculated as the sum of time taken by $n$ pre-copy iterations and pre-migration overheads (Equation 10). The number of pre-copy iterations depends upon the page modification rate of an application running on VM. In this work, it is assumed that the considered BoTs application is running with high memory page modification rate. This makes the pre-copy iterations adding maximum overheads which is equal to $n$ times of the VM size, $vm_j^{size}$ less 1 page plus pre-migration overheads.
Pre-migration overheads, \((PMO^\circ)\) are the overheads incurred before the migration starts such that resource reservation and migration initiation.

\[
MO_{ij} = PMO^\circ + \left(\frac{n \times \text{vm}_j^{size} - (P_{size})}{L_{speed}}\right)
\]  

\(\text{vm}_j^{size}\) corresponding to a VM is calculated as the product of total memory allocated to the VM, \(vm_j^{mem}\) and utilization of the VM, \(u_j\). Downtime overhead, \(DTO_{ij}\) for a task \(t_i\) of VM, \(vm_j\) is calculated as the sum of time taken to migrate the entire copy of the image of a VM and post-migration overheads, \(PMO^\sim\) (Equation 11). Post-migration overheads are the overheads incurred during the re-activation of a migrated VM.

\[
DTO_{ij} = \left(\frac{\text{vm}_j^{size}}{L_{speed}}\right) + PMO^\sim
\]

It is assumed that \(PMO^\circ\) and \(PMO^\sim\) are independent from the VM size and link speed, \(L_{speed}\). So the overhead values remained constant during each VM migration. The proposed VM migration model is applicable to VM consolidation as well.

5.2. Task Finishing Time with VM Migration Overheads

When a VM, \(vm_j\) running on a physical host, \(n_k\) executing a set of tasks \(\Gamma_j\) gets migrated, the length of each task in the set \(\Gamma_j\) changes because of the migration overheads. Apart from the migration overheads, occurrence of failures also impact the execution time or finishing time a task. \(T^*\) represents the time required to re-execute the part of a task, which is equal to the part of task length executed before the occurrence of a failure. Besides the migration overheads and re-execution part of a failed task, time to return (TTR) from a failed state to running state also adds to the finishing time of a task. So the finishing time of a task \(t_i\), of length \(l_i\) executing on a \(vm_j\) with \(n\) migrations and \(m\) failures is calculated as follows

\[
F_{ij} = \begin{cases} 
 l_i + \sum_{p=0}^{n} TMO_{ij, p} + \sum_{q=0}^{m} TTR_{ij, q} + \sum_{q=0}^{m} T^*_{ij, q}, & \text{if } p, q > 0 \\
 l_i, & \text{otherwise}
\end{cases}
\]

In the absence of VM migration, the only factors contributing to the finishing time of a task are the re-execution part of a task \((T^*)\) and time to return (TTR) from the failed state to the working state.
6. VM Checkpointing based on Failure Prediction

As a reactive fault tolerance mechanism, checkpointing is adopted in this work. Fault tolerance obtained by adopting checkpointing comes with huge cost in terms of task execution overheads. For instance, checkpointing adds the overhead of 151 hours for a job of length 1000 hours in a petaflop systems [33]. However, in the case of failure occurrence, re-execution of the failed task from the last checkpointing saves good amount of time. In general, events to create and save checkpoints get triggered with regular intervals. However, in this work checkpoints are created according to the failure prediction results in order to optimize the checkpointing overheads and to bring the evaluation in accordance with VM migration. In order to further optimize the checkpointing overheads, risk based checkpointing is used such that if the expected amount of lost work before the checkpoint is smaller than the cost of checkpoint then skip the checkpoint [? ].

6.1. VM Checkpointing Overheads

Similar to VM migration, the checkpointing overheads, $T_{ij}''$ for a task, $t_i$ executing on VM, $vm_j$ varies according to its utilization, $u_j$. The overheads are calculated as the product of maximum overhead imposed by a checkpoint, $CO_{max}$ and VM utilization.

$$T_{ij}'' = CO_{max} \times u_j$$  \hspace{1cm} (13)

6.1.1. Task Finishing Time with VM Checkpointing

When a checkpointing event gets triggered for a VM, $vm_j$ executing a set of tasks $\Gamma_j$, the current state such that the executed length ($T^#$) of all
the running tasks gets saved as a backup (Figure 6). The storage where all the backups are stored is assumed to be failure free. While saving the checkpoints, the length of all the tasks in the set $\Gamma_j$ changes because of the checkpointing overheads (Equation 13). Besides the checkpointing overheads, other major factors that contribute to the finishing time of a task is the re-execution of the lost part of the task. $T^*$ represents the time required to re-execute the lost part of a task from the last checkpoint because of a failure. Besides the checkpointing overheads and re-execution part of the failed task, time to return (TTR) from the failed state to running state also adds to the finishing time of a task. So the finishing time of a task $t_i$ of length $l_i$, executing on a $vm_j$ with $n$ checkpoints and $m$ failures is calculated as follows

$$F_{ij} = \begin{cases} l_i + \sum_{p=0}^{n} T''_{(ij)p} + \sum_{q=0}^{m} TTR_{(ij)q} + \sum_{q=0}^{m} T^*_{(ij)q}, & \text{if } n, m>0 \\ l_i & \text{otherwise} \end{cases} \quad (14)$$

7. Energy Consumption Model

As the execution time of a running task changes because of failures, VM migration and checkpointing overheads (Equation 12-14), energy consumption by the system while executing the task get affected. The energy consumption by a VM, $vm_j$ with utilization $u_j$ running on a failure prone node $n_k$ while executing a task $t_i$ of length $l_i$ is calculated as the sum of the energy consumed to execute the actual length of the task and energy wasted to execute the overheads.

$$E_{vm_{ij}} = (P_k (u_j) \times l_i) + E_{waste_{ij}} \quad (15)$$

But all the overheads that increase the execution time of a task do not contribute to the energy wastage. Among the factors such as $TMO$, $T''$, $TTR$ and $T^*$ that are considered to formulate the task finishing time in Equation 12 and 14, $TTR$ did not contribute to the energy wastage because during the down-time, a system is in non-working state. There are many other precise details that are taken into consideration in the following subsections to formulate the energy wastage in order to predict the actual energy consumption.
7.1. Energy Wastage with VM Migration Overheads

To calculate the energy wastage while using VM migration as a fault tolerance mechanism, the energy wastage is further split into two parts i.e. energy wastage due to VM migration overheads and energy consumption to re-execute the lost part of the task because of failures. According to Equation 9, total migration overheads are the sum of migration overheads, $MO$ and downtime overheads, $DTO$. $MO$ does not contribute to the energy wastage, as during the migration, VM is in transition state and not running on any physical machine. So energy wastage is calculated as

$$E_{waste_{ij}} = E_{DTO_{ij}} + E_{ij}^*$$

(16)

But downtime, $DTO$ is different from the downtime faced by a VM due to the occurrence of a failure i.e. $TTR$. During this downtime, a VM gets migrated to some other provisioned node with enough idle resources or a new node gets provisioned to accommodate the VM. During this process, an idle CPU core with 10% or less utilization, $u_{idle}$ gets provisioned but not activated. The resource gets activated once the VM migration is completed. So the duration between after provisioning and before activation of a CPU core with $u_{idle}$ is considered as the contributor to $E_{waste}$ and is calculated as follows

$$E_{DTO_{ij}} = \begin{cases} 
(P_k (u_{idle})) \times \sum_{p=0}^{n} DTO(ij)p, & \text{if } n>0 \\
0, & \text{otherwise} 
\end{cases}$$

(17)

7.2. Energy Wastage with Checkpointing Overheads

For checkpointing, energy wastage for a VM, $vm_j$ running on a node, $n_k$ while executing a task, $t_i$ is split into the energy consumption while saving the checkpoints and energy consumption while re-executing the lost part of a task.

$$E_{waste_{ij}} = E_{T''_{ij}} + E_{ij}^*$$

(18)

The power consumption while creating and saving checkpoints $P_{chkpt}$ is found to be higher by 9 to 11% than the idle power $P(u_{idle})$ and much lower than the power consumption during the execution of a task [? ] [34]. This is because during the creation of checkpoints the activity of CPU decreases (biggest energy consumer) and activity of I/O controllers i.e. Direct Memory Access (DMA) controller increases in order to perform read/write operations
on backup drives. So the energy wastage for task $t_i$ running on a VM $vm_j$ while using checkpointing is calculated as follows

$$E_{T''_{ij}} = \begin{cases} (f_{chkpt} \times P_k(u_{idle})) \times \sum_{p=0}^{n} T''_{(ij)p}, & \text{if } n>0 \\ 0, & \text{otherwise} \end{cases}$$

(19)

where $f_{chkpt}$ is the fraction of idle power consumed during a checkpointing operation. For both VM migration and VM checkpointing, the energy consumption while re-executing the lost part of a task due to the occurrence of $m$ failures is calculated as follows

$$E^*_{{ij}} = \begin{cases} P_k(u_j) \times \sum_{q=0}^{m} T^*_{{(ij)q}}, & \text{if } m>0 \\ 0, & \text{otherwise} \end{cases}$$

(20)

By using Equation 20, energy wastage without VM migration and VM checkpointing can also be calculated because the only overhead that contributes to the energy wastage while not using VM migration or VM checkpointing is the task re-execution part. So the total energy consumption by $x$ provisioned nodes allocated to $m$ VMs while finishing all the tasks of BoT application in the presence of the occurrence of failures, using VM migration and VM checkpointing as the fault-tolerance mechanisms is calculated as

$$E_{total} = \sum_{k=1}^{x} \sum_{j=1}^{m} E_{vm_{jk}}$$

(21)

8. Resource and VM Management

Given the set of tasks and failure prone resources, the problem is how to provision the resources and allocate the VMs executing the tasks to maximize the reliability and minimize the energy consumption while keeping the number of provisioned resources minimum and ensuring every task to complete before the deadline.

Before provisioning the physical resources from the pool of resources, $P$, the first challenge was to calculate the number of VMs need to be instantiated to execute the tasks. Problem of allocation of tasks to VMs is formulated as a bin-packing problem [35] where the VM is considered as a bin and capacity of a bin is the utilization level of VM. Each task, $t_i$ of a BoTs $B$
## Table 1: Nomenclature used in algorithms

<table>
<thead>
<tr>
<th>Notation</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_i$</td>
<td>$i_{th}$ task</td>
</tr>
<tr>
<td>$l_i$</td>
<td>Length of $i_{th}$ task</td>
</tr>
<tr>
<td>$l_{max}$</td>
<td>Length of a longest task in $T$</td>
</tr>
<tr>
<td>$r_k$</td>
<td>$k_{th}$ node</td>
</tr>
<tr>
<td>$\mathcal{R}$</td>
<td>List of Provisioned nodes, $r_k \in \mathcal{R}$</td>
</tr>
<tr>
<td>$vm_j$</td>
<td>$j_{th}$ virtual machine</td>
</tr>
<tr>
<td>$u_j$</td>
<td>Utilization corresponding to each VM</td>
</tr>
<tr>
<td>$C^*_h$</td>
<td>Number of Idle cores on a target host, $H$</td>
</tr>
<tr>
<td>$C^*_k$</td>
<td>Number of Idle cores on a destination host, $R_k$</td>
</tr>
<tr>
<td>$\Psi_k$</td>
<td>Ratio of $MTBF_k$ and $P_k$</td>
</tr>
<tr>
<td>$T_{h\sim}$</td>
<td>Next Prediction Time for Target Host, $H$</td>
</tr>
<tr>
<td>$T_{k\sim}$</td>
<td>Next Prediction Time for Destination Host, $R_k$</td>
</tr>
<tr>
<td>$rel_{jk}$</td>
<td>Reliability of $j_{th}$ virtual machine on $k_{th}$ node</td>
</tr>
<tr>
<td>$pow_{jk}$</td>
<td>Power consumption of $j_{th}$ virtual machine on $k_{th}$ node</td>
</tr>
</tbody>
</table>

had a corresponding utilization, which is calculated by normalizing the task length, $l_i$ w.r.t to the task of maximum length $l_{max}$ in $B$. The minimum number of VMs, $N_{min}$ required to instantiate to allocate all the $n$ tasks of $B$ is calculated as follows

$$N_{min} = \left\lceil \frac{\sum_{i=1}^{n} l_i}{l_{max}} \right\rceil \quad (22)$$

Before instantiating new VMs, VM provisioning and task allocation algorithm (Algorithm 1) checks the available capacity on the running VMs and tried to allocate maximum number of tasks to the running VMs (step 6-16). For the remaining tasks, minimum number of new VMs need to be instantiated is calculated (step 19-23). After calculating the minimum number of VMs, optimized allocation of remaining tasks is done using Best Fit Bin Packing algorithm [35] while taking the status of all the running VMs into consideration.

After calculating the minimum number of VMs, physical machines from the pool of heterogeneous resources are provisioned (Algorithm 2). The physical resources in the pool have different hazard-rates and power profiles and are chosen on the basis of the objective of the cloud provider to minimize the
Algorithm 1: VM Provisioning and Task Allocation

**Input:** Bag of Tasks, \( B \) and List of VMs, \( V \)

**Output:** Set of Provisioned VMs and Allocated Tasks

1 //Calculating the normalized length of each task in \( B \);
2 for \( i \in B \) do
3 \[ n_i = l_i / l_{\text{max}} ; \]
4 \[ N \leftarrow n_i ; \]
5 //Allocating tasks to currently running VMs;
6 for \( i \in B \) do
7   for \( j \in V \) do
8     if \( n_i == 1.0 \) then
9       break ;
10      \( u_j \leftarrow \text{vm}_j\.currentUtilization() ; \)
11      tempUtli = \( u_j + n_i ; \)
12      if \( \text{tempUtli} \leq 1.0 \) then
13        \( t_i \leftarrow \text{vm}_j\.allocateVM(B) ; \)
14        \( u_j = u_j + n_i ; \)
15        \( \text{vm}_j \leftarrow u_j\.setUtilization() ; \)
16        break ;
17 //For unallocated tasks, new VMs gets instantiated;
18 \( B \leftarrow B\.unAllocatedTasks() ; \)
19 //Calculate the minimum no. of VMs using Equation 21;
20 \( N_{\text{min}} \leftarrow \text{getVMCount()} ; \)
21 //Provision physical resources and allocate new VMs (Algorithm 2);
22 for \( k \in N_{\text{min}} \) do
23  \( \text{vm}_k \leftarrow \text{AllocatePhysicalResources()} ; \)
24 //Allocate unallocated tasks to new VMs;
25 Goto 6

xxii
energy consumption or maximize the reliability or achieve both objectives at the same time. As the focus of this work is to maximize the reliability and minimize the energy consumption together, Reliability-Energy Aware Best Fit Decreasing (REABFD) resource provisioning and VM allocation policy (function Reliability and Energy Aware Best Fit Decreasing) is chosen to rank the physical resources by using the ratio of MTBF and power consumption corresponding to current utilization of each resource. Once physical resources are provisioned and allocated to instantiated minimum number of VMs, an optimized allocation of remaining tasks is done while taking the status of all the running VMs into consideration (Algorithm 1).

<table>
<thead>
<tr>
<th>Function</th>
<th>Reliability and Energy Aware Best Fit Decreasing (REABFD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 function ReliabilityAndEnergyAware(R)</td>
<td></td>
</tr>
<tr>
<td>2 for k ∈ R do</td>
<td></td>
</tr>
<tr>
<td>3 [ MTBF_k ← r_k.calculateMTBF(); ]</td>
<td></td>
</tr>
<tr>
<td>4 [ P_k ← r_k.calculatePowerConsumption(); ]</td>
<td></td>
</tr>
<tr>
<td>5 [ \Psi_k ← (MTBF_k)/(P_k); ]</td>
<td></td>
</tr>
<tr>
<td>6 for k ∈ R do</td>
<td></td>
</tr>
<tr>
<td>7 [ R_{sorted} ← \Psi_k.sortMTBFPowerRatioIncreasing(); ]</td>
<td></td>
</tr>
<tr>
<td>8 return R_{sorted};</td>
<td></td>
</tr>
</tbody>
</table>

While allocating VMs to physical machines, RMS first checks the availability of idle cores on the provisioned and active physical machines ∈ ℜ and allocate VMs to the idle cores, if available (step 4-17). If the resource requirement for a VM, \( vm_j \) is not fulfilled by physical machines ∈ ℜ, then a new physical machine is provisioned from \( P \) (step 18). To tolerate or avoid the occurrence of failures, failure prediction is used in order to trigger the fault tolerance mechanisms. Once the fault tolerance mechanism either reactive or proactive is triggered for a node, the node is labelled as predictedtofail and was not available for the allocation to new or migrating VMs (Algorithm 3) until the label is removed (step 8). The label is removed after the predicted failure is occurred. In the case of checkpointing, VM remains on the same node throughout its lifetime. On the basis of the failure prediction results, the present state of all the running tasks on VM gets saved so that in case of a failure, they will get recovered from the last checkpoint. The failing node is
Algorithm 2: Resource Provisioning and VM Allocation

**Input:** List of Resources, \( R \) and List of VMs, \( V \)

**Output:** Set of Provisioned Resources and Allocated VMs

1. \( R_{\text{sorted}} \leftarrow \text{RELIABILITYANDENERGYAWARE}(R); \)
2. for \( j \in V \) do
3.   \( VM_\text{cores}_j \leftarrow \text{vm}_j.\text{coresRequired}(); \)
4.   for \( k \in R \) do
5.     if \((R_k == \text{failed})\) then
6.       continue;
7.     else
8.       if \((R_k.\text{predictedtoFail}()! = \text{true}) \land (R_k \geq VM_\text{cores}_j)\) then
9.         \( r_k \leftarrow \text{vm}_j.\text{allocateHost}(); \)
10.        \( RC_k = RC_k - VM_\text{cores}_j; \)
11.       //CALCULATE VM RELIABILITY (EQUATION 5);
12.        \( R_{\text{vm}, j} \leftarrow \text{vm}_j.\text{calculateReliability}(); \)
13.       //ESTIMATE VM POWER CONSUMPTION (EQUATION 7);
14.       \( \text{pow}_k(u_j) \leftarrow \text{vm}_j.\text{estimatePower}(); \)
15.       if \((RC_k == 0)\) then
16.         \( R_{\text{sorted}} = R_{\text{sorted}} - R_k; \)
17.       break;
18.     
19. //IN CASE OF UNALLOCATED VM, PROVISION NEW HOST FROM POOL, \( P \)
20. if \((\text{vm}_j.\text{unallocated}() == \text{true})\) then
21.   \( R \leftarrow R_{\text{sorted}}; \)
22.   Goto 4

After finishing all the allocated tasks, a VM gets destroyed. Its corresponding resources sit idle and contribute to energy wastage unless these resources are allocated to a new VM. In order to reduce such idle resources, VM consolidation (Algorithm 4) is adopted, which get triggered when a VM is destroyed. The proposed method is a failure-aware VM consolidation pol-
Algorithm 3: Failure Aware VM Migration

Input: Expected to be Failed Resource, \( R \)
Input: List of Resources, \( R \) and List of VMs, \( V \)
Output: New allocation for VMs

1 //Identify a resource with the earliest predicted failure time;
2 \( V \leftarrow R_k.VmList() \);
3 for \( j \in V \) do
4 \( R_k.deallocate(\text{vm}_j) \);
5 \( \text{vm}_j.setInMigration() \);
6 //Resource provisioning and VM allocation Algorithm will be called to
7 //select a new host for the migrating VM;
8 \( R_k.setExpectedtobeFailed(true) \);
Algorithm 4: Failure Aware VM Consolidation

**Input:** List of Resources, $\mathcal{R}$ and Target Host, $\mathcal{R}_h$

**Output:** New host and set of VMs

1. $V_h \leftarrow \mathcal{R}_h.VmList();$
2. $C^* = \mathcal{R}_h.numberofIdleCores();$
3. $T^*_{h} = \mathcal{R}_h.nextFailurePredictionTime();$
4. for $k \in \mathcal{R}$ do
5.   $T_{k}^* = \mathcal{R}_k.nextFailurePredictionTime();$
6.   $C^*_k = \mathcal{R}_k.numberofIdleCores();$
7.   $V_k \leftarrow \mathcal{R}_k.VmList();$
8.   if (($C^*_k \geq V_h.size()$) $\&\&$ ($T_{k}^* < T^*_{h}$)) then
9.     if $\mathcal{R}_k.predictedtoFail() == true$ then
10.    continue;
11.   flag == true;
12.   for $j \in V_h$ do
13.     $\mathcal{R}_h.deallocate(vm_j);$  
14.     $vm_j.setInMigration();$
15.     $vmAllocation(vm_j, \mathcal{R}_k);$  
16.     //Add overheads using Equation 12
17.   if (($C^*_h \geq V_k.size()$) $\&\&$ ($T^*_{k} < T^*_{h}$)) then
18.     if $\mathcal{R}_h.predictedtoFail() == true$ then
19.        continue;
20.     for $j \in V_k$ do
21.        $\mathcal{R}_k.deallocate(vm_j);$  
22.        $vm_j.setInMigration();$
23.        $vmAllocation(vm_j, \mathcal{R}_h);$  
24.        //Add overheads using Equation 12
25.     if (flag == true) then
26.        //Turning off the target host;
27.        setState($\mathcal{R}_h, off$);
28.        $\mathcal{R} = \mathcal{R} - \mathcal{R}_h;$  
29.        break;
30.    else
31.        //Turning off the destination host;
32.        setState($\mathcal{R}_k, off$);
33.        $\mathcal{R} = \mathcal{R} - \mathcal{R}_k;$  
34.        break;

Table 2: Simulation Configuration Parameters

<table>
<thead>
<tr>
<th>Input Parameters</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stringency Factor ($f$)</td>
<td>1.3</td>
</tr>
<tr>
<td>Sensitivity Factor ($\beta$)</td>
<td>1</td>
</tr>
<tr>
<td>Number of Pre-Copy Iterations ($n$)</td>
<td>4</td>
</tr>
<tr>
<td>Page Size ($P_{size}$)</td>
<td>4KB</td>
</tr>
<tr>
<td>Pre-Migration Overheads ($PMO^o$)</td>
<td>15.6403 secs</td>
</tr>
<tr>
<td>Post-Migration Overheads ($PMO^\sim$)</td>
<td>.9070 secs</td>
</tr>
<tr>
<td>Link Speed ($L_{speed}$)</td>
<td>1Gbps</td>
</tr>
<tr>
<td>Smoothing Constant ($\alpha$)</td>
<td>.9</td>
</tr>
<tr>
<td>Maximum Checkpointing Overhead ($CO^{max}$)</td>
<td>20 secs</td>
</tr>
<tr>
<td>Idle Power Checkpointing Fraction ($f_{chkpt}$)</td>
<td>1.15</td>
</tr>
</tbody>
</table>

9. Performance Evaluation

After doing a Matlab based prototyping of all the proposed mathematical models and algorithms, a simulation based evaluation of the same has been done using a real cloud computing architecture purposed in figure 1 and using configuration parameters given in table 2. In order to simulate the architecture, we have extended a known cloud computing simulator ’CloudSim’ [36] by adding failure injector, fault tolerance and VM consolidation.

9.1. Datacenter Configuration

Grid5000 failure dataset collected for 1.5 years between 2005-2006 is used in this work to retrieve information about the failures and hardware configuration of approximately 1300 hosts of the datacenter. The failure traces have been downloaded from Failure Trace Archive (FTA)[21], an online public repository providing failure traces gathered from 26 different computing sites. This work has used Grid5000 traces specifically because of the precise details provided regarding the failure start time and end time. The mean time between failures (MTBF) and mean time to return (MTTR) for each node at maximum utilization have been calculated by using the failure information provided in the traces.

In order to choose the value of smoothing constant for the failure prediction (Equation 8), a statistical analysis of failure accuracy is conducted using different values of smoothing constant (Figure 7). The failure prediction accuracy is calculated as the percentage of failures predicted before the
occurrence of failures. The analysis is conducted using nodes from each cluster of 9 different sites with the maximum number of failure events. From the analysis, it is observed that as the value of smoothing constant increases from 0.2 to 0.9, the failure prediction accuracy also increases. Consequently, the less number of past values contribute to the short term prediction, the better the prediction results are achieved. Same behaviour is observed by using moving average prediction method such that by using smaller window size better failure prediction accuracy is achieved. This is because of the interpolation prediction being performed such that for each failure event value in failure traces, a corresponding failure prediction value is generated. If the generation of failure prediction values beyond the number of values provided in failure traces (extrapolation) was required then the contribution of past prediction values (smaller smoothing constant value or larger window size) would have been more desirable. By using exponential smoothing with smoothing constant 0.9, the achieved prediction accuracy is between 57% to 71%.

For pre-migration, $PMO^o$ and post-migration, $PMO^-$ overheads, see appendix 11.1 and appendix 11.2, respectively. To keep the deadlines corresponding to tasks moderate [19] and to keep a strict linear relationship [11] between the utilization and reliability, values for stringency factor, $f$ and sensitivity factor, $\beta$ are set to 1.3 and 1, respectively. Idle power checkpointing
fraction, $f_{\text{chkpt}}$ is set to 1.15 [34] and other parametric values for VM migration overheads are obtained from [32]. To calculate the power consumption, the values of minimum and maximum power consumptions corresponding to a node are taken from spec2008 benchmark\(^3\). To select the realistic datacenter nodes, we have matched the core count and memory capacity of the nodes with the values provided in the traces. On the basis of information, we have selected Intel Platform SE7520AF2 Server Board, HP ProLiant DL380 G5 and Dell PowerEdge R710 as 2, 4 and 8 cores with 4GB, 16GB and 12GB memory nodes, respectively. To generate the BoTs workload, model proposed by Iosup et al. [37] is used with parameters given in Table 3.

### 9.2. Workload Model

To generate the BoTs workload, model proposed by Iosup et al. [37] has been used with parameters given in table 3. To provision the enough number of nodes for the fair evaluation of proposed policies, the inter-arrival time has been modelled using peak time workload following Weibull distribution. Every incoming BoT consists of $2^x$ tasks where $x$ follows Weibull distribution with scale and shape parameters given in table 3. The length or execution time of each task in a BoT has been modelled as normal distribution. Every task has a corresponding deadline that is calculated using equation 1.

### 9.3. Performance Evaluation Metrics

To evaluate the performance of the proposed resource provisioning and VM allocation policies, the results of following metrics have been reported

1. **Reliability**: The reliability with which the application has been executed on the provisioned resources.

\(^3\)https://www.spec.org/power_ssj2008/results/

---

Table 3: Workload Generation Parameters

<table>
<thead>
<tr>
<th>Input Parameters</th>
<th>Distribution</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inter-arrival time (BoT)</td>
<td>Weibull</td>
<td>Scale = 4.25, Shape = 7.86</td>
</tr>
<tr>
<td>Number of Tasks per BoT</td>
<td>Weibull</td>
<td>Scale = 1.76, Shape = 2.11</td>
</tr>
<tr>
<td>Average runtime per task</td>
<td>Normal</td>
<td>Mean = 2.73, SD = 6.1</td>
</tr>
</tbody>
</table>
2. **Number of Failures**: The total number failures occurred while executing the application.

3. **Completion Time**: It is the time taken by each task of BoT application to finish.

4. **Re-execution Time**: Time to re-execute the lost part of a task because of the occurrence of failures.

5. **Down Time**: It is the down time that has been faced by the application because of the occurrence of failures.

6. **Application Utility Value**: The computation value (equation 2) of the application that has been achieved despite of deadline violations.

7. **Energy Consumption**: Energy consumption incurred by the provisioned resources while executing the application.

8. **Energy Wastage**: The amount of energy wasted while re-executing the lost of part of the task because of failures and related overheads.

9. **Idle Energy Consumption**: Energy consumed by provisioned resources sitting idle such that no tasks are being processed by the resources.

10. **Benefit Ratio**: It is the the ratio of achieved reliability and energy consumption.

### 9.4. Results and Discussion

Performance evaluation of the proposed resource provisioning and VM consolidation policy using different fault tolerance mechanisms is done in terms of reliability, finishing time and energy-efficiency. All the simulations are performed using 200 BoTs consisting of total number of tasks between 20000 to 25000. All the reported results are the average of 50 simulations with 95% confidence interval. For brevity, in the discussion of results below, 'Rstr' represents the base scenario of 'restart' in which no fault tolerance mechanism is used, and Chkpt, Mig and Mig/Chkpt represents scenarios with checkpointing, VM migration and combination of VM migration and checkpointing, respectively. **Note**: VM migration is used for two purposes such that for fault tolerance and for energy regulation (VM consolidation). Though the method is same, purpose and triggering criteria are different. In the case of fault tolerance (Algorithm 3), VM migration get triggered on the basis of failure prediction results. However, VM consolidation (Con) (Algorithm 4) get triggered when a VM is finished the execution of its corresponding tasks and get destroyed and corresponding resources become idle.
9.4.1. Reliability Evaluation

Figure 8a presents the average reliability of the system using various fault tolerance mechanisms with and without consolidation. For all the scenarios, it is observed that despite of the extra migration overheads imposed while performing VM consolidation, the system possessed higher reliability than the scenarios without consolidation. Which is achieved despite of the fact that by imposing extra overheads, task length increases which in return reduces the reliability of a system. However, the obtained results are because of a huge reduction of approximately 73% in the occurrence of failures (Figure 8b) for the scenarios using failure-aware VM consolidation in comparison to without VM consolidation scenarios. With the reduction of number of failures, the re-execution part of failed tasks is reduced significantly (Figure 8b), which complemented the VM consolidation overheads and increased the reliability of the system. Among the used fault tolerance mechanisms, it is found that VM migration (Mig) used as fault tolerance mechanism outperformed checkpointing (Chkpt) by reducing the failure count by up to 80%. This failure count is further reduced with adoption of ’Mig’ in conjunction of ’Chkpt’ (Mig/Chkpt) by 14% and 18% for both with and without consolidation environments, respectively. Such reduction of number of failures lead to significant changes in the reliability where the combination of VM migration with checkpointing used in failure-aware VM consolidation environment ensured maximum reliability among all the scenarios. **Note:** Although by
using 'Mig' and 'Mig/Chkpt', attempt is made to avoid the occurrence of failures by migrating running VMs from suspicious nodes predicted to be failed to other healthy nodes, failures still happened because of the failure prediction errors.

9.4.2. Execution Time Evaluation

Figure 9a shows the average turnaround time that is achieved while executing the tasks of BoT application. From the figure it is observed that the Rstr scenario experienced maximum task completion time for both with and without consolidation because of higher downtime (Figure 9c) and re-execution time (Figure 9b) due to large number of failures (Figure 8b).
Among the scenarios using fault tolerance, Mig/Chkpt scenario has minimum completion time such that lower by 12% and 58% from Mig and Chkpt, respectively because of less re-execution and downtime overheads. This completion time further improved by 12% with the adoption of failure-aware VM consolidation policy despite of the extra migration overheads imposed while performing consolidation. This is because of the improvement obtained in terms of number of failures which eventually reduced the re-execution time and downtime and resulted in the reduction of the turn around time corresponding to running tasks.

The objective behind reducing the turnaround time is to finish the tasks before the corresponding deadlines (Equation 1) in order to achieve maximum utility value (Equation 2) and to ensure high quality of services (QoS). Figure 9d shows the achieved application utility value while using different fault tolerance mechanisms with and without VM consolidation. From the figure, it can be seen that maximum application utility value is achieved by using the combination of VM migration and checkpointing (Mig/Chkpt). This is because if a failure happened for Mig/Chkpt scenario, the re-execution of a failed task started from the last saved checkpoint rather than starting from the beginning, which is happening while using only VM migration (Mig) as fault tolerance mechanism. Starting from the last checkpoint in ‘Mig/Chkpt’ scenario resulted in the lowest downtime (Figure 9c) and re-execution time (Figure 9b). An improvement of 7% over Mig and 27% over Chkpt is achieved without using consolidation. By introducing failure-aware VM consolidation the utility value is further improved by 1% for Mig, 2% for Mig/Chkpt and 7% for Chkpt. Such improvement of utility value shows that despite of extra migration overheads imposed while performing VM consolidation, more deadlines are achieved than without consolidation with a good improvement over the energy wastage and reliability of the system.

9.4.3. Energy Efficiency Evaluation

Figure 10a presents the average energy consumption of the system. Obviously, system has consumed maximum energy under ‘Rstr’ because of the large re-execution overheads (Figure 9b) incurred due to the occurrence of large number of failures (Figure 8b). However, among the scenarios with fault tolerance mechanisms, ‘Chkpt’ consumed maximum energy such that while using consolidation it is more by 26% from Mig and 34% from Mig/Chkpt and for without consolidation the increment is 50% from Mig and 56% from Mig/Chkpt. This is because of the reactive behaviour of checkpointing mech-
(a) Average Energy Consumption (kWh)

(b) Average Energy Wastage

(c) Average Idle Energy

(d) Benefit Ratio

Figure 10: Results for Energy Efficiency Evaluation
anism where the occurrence of failures is certain and recovery happens after the occurrence of a failure and to tolerate the impact of a failure, execution restarts from the last checkpoint. However, in VM migration based scenarios, the attention is also paid on the failure avoidance besides the failure tolerance. Which resulted in lesser re-execution time and system downtime because of lower failure count. Among the Mig and Mig/Chkpt scenarios, less energy consumption is experienced for Mig/Chkpt scenario for both with and without consolidation. In all the cases, scenarios using consolidation found to be using less energy consumption than scenarios without consolidation despite of extra VM migration overheads. This is because by using the proposed failure-aware VM consolidation policy (Algorithm 4), less failure occurrence is experienced (Figure 8b) which in return reduced the re-execution time (Figure 9b) of tasks.

The main objective of using VM consolidation is to reduce the number of underutilized resources by turning them off or by putting them on sleep/hibernation mode after migrating the corresponding VMs to other underutilized resources. By doing this, the resource efficiency and utilization increases and reduces the idle energy consumption by either idle resources or underutilized resources. Figure 10c shows a significant reduction in idle energy consumption of the system while performing the proposed failure-aware VM consolidation policy. Such that, the idle energy consumption is brought down by 86% for Chkpt, 67% for Mig/Chkpt, 62% for Mig and 78% for Rstr. Among all the fault tolerance scenarios, Mig/Chkpt consumed minimum idle energy which is lesser by 23% from its nearest rival scenario i.e. scenario using Mig because of lesser failure count (Figure 8b) and turnaround time (Figure 9a). Lower the turnaround time, early the tasks will finish the execution, which will proceed to turning the idle or underutilized resources off earlier.

As the objective of this work is to improve the reliability and energy efficiency jointly, a metric called Benefit Ratio (the ratio of achieved reliability and energy consumption) is introduced to reflect the overall improvement of proposed mechanisms (Figure 10d). While being consistent with previous results, the combination of ‘Mig’ and ‘Chkpt’ gives the best benefit ratio value which is higher by 23% than the scenario using ‘Mig’ as the fault tolerance method. This value is further improved by 15% with the adoption of proposed failure-aware VM consolidation policy.
10. Conclusion

In this paper, a failure-aware energy-efficient VM consolidation method is presented. It takes the reliability factor into consideration before consolidating the running VMs in order to save energy in a failure prone cloud computing environment. To provide fault tolerance, both reactive (checkpointing) and proactive (VM migration) mechanisms are used. To trigger the fault tolerance mechanisms, time series analysis based failure prediction is introduced. Verified by our extensive simulation study, the following conclusions are drawn.

1. While performing the VM consolidation in a failure prone cloud computing system, a significant improvement in terms of energy efficiency and reliability can be achieved by considering the failure characteristics of physical resources.

2. To achieve higher fault tolerance in cloud computing systems, it is better to use the combination of reactive and proactive fault tolerance mechanisms rather than using them individually.

In future work, we will explore the failure correlation and new methods will be designed to provide fault tolerance in an energy efficient manner in the presence of correlated failures. Proposed failure-aware VM consolidation method will be tested in the presence of correlated failures.

11. Appendix

Sherif et al. [32] has been advised that with a VM of 1,024 MB memory and 1 Gbps migration link, migration time is equal to 50 seconds and downtime is .314. By using the given values, $PMO^{o}$ and $PMO^{-}$ have been calculated as follows.

11.1. Calculation of Pre-Migration Overheads

\[
50 = PMO^{o} + \left( \frac{((4 \times (1024^{3} \times 8)) - (4 \times 1024 \times 8)))}{1000^{3}} \right)
\]

\[
50 = PMO^{o} + 34.3597
\]

\[
PMO^{o} = 15.64 \text{ seconds}
\]
11.2. Calculation of Post-Migration Overheads

\[ 9.497 = PMO^\sim + \left( \frac{1024^3 \times 8}{1000^3} \right) \]

\[ PMO^\sim = 0.9070 \text{ seconds} \]

References


[5] S. Anthony, Microsoft now has one million servers—less than google, but more than amazon, says ballmer, ExtremeTech. ExtremeTech 19.


